EACA Colour Genie EG2000

Anwenderhandbuch

Vorwort

Herzlichen Glückwunsch zu Ihrem neuen Colour-Genie Computer! Mit dem Colour-Genie haben Sie einen Computer erworben, der Ihnen durch seine hochmoderne Technik Möglichkeiten bietet, die noch vor wenigen Jahren nur für viele tausend Mark zu haben waren.

Durch dieses Handbuch werden Sie Schritt für Schritt immer neue Fähigkeiten Ihres Computers kennen und beherrschen lernen. Ich gehe dabei davon aus, daß der Umgang mit einem Computer für Sie eine neue Erfahrung ist. Daher ist dieses Handbuch so ausgelegt, daß Sie die Basic-Programmiersprache Ihres Computers „von der Pike an" lernen können. Dabei wird alle graue Theorie durch reichlich — im wahrsten Sinne des Wortes — farbige Praxis anhand von Programmierbeispielen vertieft werden. Denjenigen von Ihnen, die schon Computer-Erfahrung besitzen, oder die von einem anderen Computer zum Colour-Genie gewechselt haben (was bei dessen Preis-Leistungsverhältnis nicht selten sein dürfte), wird anfangs Einiges etwas langwierig vorkommen, aber auch sie sollten dieses Handbuch durchlesen; im fortgeschrittenen Teil werden auch sie interessante „Programmier-Leckerbissen" finden. Ich hoffe, daß Sie mit diesem Handbuch Ihren Computer beherrschen lernen, der dann ein wertvolles Werkzeug für Sie sein wird.

Sei es als Farb-Genie, Unterhaltungs-Genie, Mathe-Genie, Grafik-Genie, Musik-Genie, Spiel-Genie, Kartei-Genie, Text-Genie, Schul-Genie, Berufs-Genie . . .

. . . in jedem Falle wünschen wir Ihnen viel Spaß mit Ihrem neuen Computer!



Inhalt

1. Aufstellung und Anschluß des Colour-Genies

Nach dem Auspacken müssen Sie sich zuerst vergewissern, daß das Gerät durch den Transport keinen äußerlich sichtbaren Schaden erlitten hat.

Dann sollten Sie sich Gedanken über die Aufstellung des Computers machen. Das Colour-Genie kann an einen normalen Pal-Farbfernseher oder auch an einen SW-Fernseher angeschlossen werden. Ein Farbfernseher ist natürlich vorzuziehen. Wählen Sie den Aufstellungsplatz dort, wo Sie das Bild des Fernsehers gut einsehen können und wo Sie zugleich die Tastatur gut bedienen können.

Achten Sie ferner darauf, daß Sie die Lüftungschlitze auf der Unterseite des Computers nicht blockieren.

Der Anschluß an den Fernseher ist recht einfach: Aus dem Colour-Genie führt ein Kabel heraus, das am Ende einen Antennenstecker hat. Diesen stecken Sie in die Antennenbuchse des Fernsehers. Dann müssen nur noch die Netzstecker des Fernsehers und des Computers eingesteckt werden. Achten Sie darauf, daß diese fest sitzen.

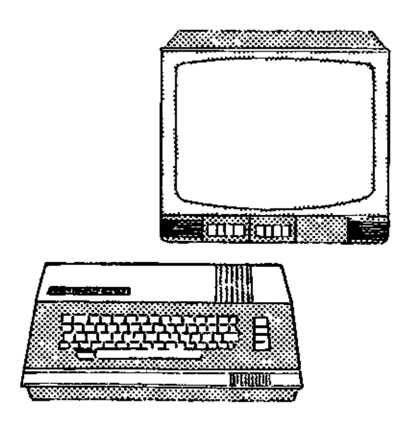
Nun schalten Sie beide Geräte ein. Der Computer hat seinen Netzschalter links auf der Rückseite. Wenn der Computer eingeschaltet ist, leuchtet die rote Leuchtdiode links neben der Tastatur auf. Abschließend muß noch der Fernseher auf den Computer abgestimmt werden. Wie Sie Ihren Fernseher abstimmen, können Sie aus der Bedienungsanleitung des Fernsehers entnehmen.

Der Computer erzeugt sein Bild auf UHF-Kanal 36 oder auf VHF-Kanal 3 — den richtigen Wert können Sie vom Aufkleber auf der Unterseite des Computers ablesen.

Wenn der Fernseher richtig abgestimmt ist, erscheint die farbige Schrift:
MEM SIZE?

Jetzt können Sie noch die Farbsättigungs-, Kontrast- und Helligkeitsregler des Fernsehers so einstellen, daß die Schrift möglichst gut lesbar ist.

Nun kann die Arbeit mit dem Computer beginnen!



2. Die ersten Schritte

Wir wollen nun folgendermaßen vorgehen: Zuerst werden Sie sehen, wie man mit dem Colour-Genie ganz normal rechnen kann. Dann werden Sie die Funktion von Variablen und anschließend die Grundzüge der Basic-Programmierung kennenlernen.

Nachdem Sie sich mit den Tastaturfunktionen Ihres Colour-Genies vertraut gemacht haben, werden die fundamentalen Basic-Befehle in aller Ausführlichkeit erklärt. Mit der Übung, die Sie dann haben werden, können Sie die restlichen Befehle und Funktionen in relativ kurzer Zeit erlernen.

Wie schon erwähnt, meldet sich der Computer nach dem Einschalten mit der Frage

MEM SIZE? ■

Diese Abfrage ermöglicht es, Speicher zu reservieren. Dies ist aber jetzt noch nicht interessant. Drücken Sie nun die <RETURN>-Taste.

Diese Taste wird immer dann gedrückt, wenn eine Eingabe abgeschlossen ist und sie nun vom Computer bearbeitet werden soll. Das Bild sieht nun so aus:

MEM SIZE?
COLOUR BASIC
READY
>■

Das Colour-Genie befindet sich jetzt im Eingabe-Modus. Dies wird durch die Meldung „READY“ angezeigt.

Das „>“-Zeichen, der sogenannte Prompt, bedeutet, daß ab hier eine Eingabe erwartet wird. Das blinkende Quadrat „■“, der sogenannte CURSOR, zeigt an, wo das nächste Zeichen, das Sie eingeben, erscheinen wird. Drücken Sie z. B. irgendeine Buchstaben-Taste und Sie sehen, wie der Buchstabe auf dem Bildschirm erscheint und der Cursor eine Stelle weiterrückt. Wenn Sie nun den Buchstaben wieder löschen wollen, drücken Sie die <←>-Taste.
Diese Taste drücken Sie immer dann, wenn Sie einen Tippfehler korrigieren wollen.
Wenn Sie die <←>-Taste mehrfach drücken, werden mehrere Zeichen gelöscht.

Noch ein Hinweis vorab: Sollten irgendwann bei der Eingabe statt der gewünschten Buchstaben Grafikzeichen erscheinen, dann haben Sie vermutlich aus Versehen die <MOD SEL>-Taste gedrückt. Drücken Sie in diesem Fall nochmals die <MOD SEL>-Taste, und Sie können wieder Buchstaben eingeben.

3. Rechnen mit dem Genie

Geben Sie Folgendes ein:

```
>PRINT 12+25
```

und drücken dann die <RETURN>-Taste. Der Computer gibt das Ergebnis richtig an:

```
 47
READY
>■
```

Sehen wir uns Ihre Eingabe einmal genau an.

Zuerst das Wort „PRINT“: Dieses Wort, genauer gesagt dieser BEFEHL, bedeutet, daß der Computer alles Folgende berechnen und dann das Resultat ausgeben soll.

Die <RETURN>-Taste besagt, daß die Eingabe nun bearbeitet werden soll. Da diese Taste hinter jeder kompletten Eingabe gedrückt werden muß, wird sie im folgenden nicht jedesmal extra erwähnt werden.

Bevor wir uns mit den anderen Rechenarten beschäftigen, will ich Sie noch auf eine Vereinfachung hinweisen:

Da der „PRINT“-Befehl sehr oft gebraucht wird, kann man ihn durch ein einfaches „?“ ersetzen. Das „?“ erhalten Sie, indem Sie, wie bei einer Schreibmaschine, die <SHIFT>-Taste gedrückt halten und dabei gleichzeitig die <?>-Taste drücken.

Folgende zwei Befehle sind also vollkommen gleichbedeutend:

```
>PRINT 2+237
  239
READY
>■
```

```
>? 2+237
  239
READY
>■
```

Nun aber zu den anderen Rechenarten:

Subtraktion

Geben Sie Folgendes ein (Achtung: verwechseln Sie auf keinen Fall die Null (0) mit dem Buchstaben (O)!:

```
>PRINT 12—20
—8
READY
>■
```

Multiplikation

Multipliziert wird mit dem Stern-Zeichen (★):

```
>PRINT 7★230
 1610
READY
>■
```

Division

Dividiert wird mit dem Querstrich (/):

```
>PRINT 20 / 8
 2.5
READY
>■
```

Wenn Sie genau hinsehen, bemerken Sie, daß obiges Ergebnis nicht ein Komma, sondern einen Punkt als Trennungszeichen enthält. Dies entspricht der amerikanischen Schreibweise. Das Gleiche gilt bei der Eingabe von Kommazahlen — auch hier muß das Komma durch einen Punkt ersetzt werden.

Folgendes Beispiel ist also richtig:

```
>PRINT 12.5 / 2.5
 5
READY
>■
```

Zur Division muß noch eins gesagt werden: Das „/"-Zeichen bezieht sich nur auf die unmittelbar benachbarten Zahlen; es ersetzt nicht einen langen Bruchstrich!

```
>PRINT 3+4 / 5+7
```

entspricht also dem Term

$$3 + \frac{4}{5} + 7$$

Wollen Sie aber den Term

$$\frac{3 + 4}{5 + 7}$$

berechnen, müssen Sie Klammern setzen.

Das sieht dann so aus:

```
>PRINT ( 3+4 ) / ( 5+7 )
```

Potenzieren

Geben Sie folgenden Befehl ein:

```
>PRINT 4 [ 3
 64
READY
>■
```

(4^3 = 4 [3 = 4★4★4 = 64)

Das Potenzierungszeichen „ [“ erhalten Sie durch Drücken der Aufwärtspfeiltaste <↑>.

Die Potenz, in obigem Beispiel 3, wird nicht wie sonst üblich als Hochzahl eingegeben, da dies auf dem Bildschirm schwer darzustellen wäre, sondern das „[“-Zeichen legt die folgende Potenz fest.

Durch das Potenzieren kann man auch Wurzeln ziehen, denn es gilt:

$\sqrt[n]{X} = X^{\frac{1}{n}}$ z. B.: $\sqrt[3]{27} = 27^{\frac{1}{3}} = 3$

Computerbefehl:

```
>PRINT 27 [ ( 1 / 3 )
 3
READY
> ■
```

(Für die Quadratwurzel gibt es allerdings einen eigenen Befehl.)

Im vorigen Beispiel mußte wieder eine Klammer gesetzt werden, um das richtige Ergebnis zu erhalten.

Der Befehl ohne Klammer:

```
>PRINT 27 [ 1 / 3
```

hätte nämlich

$\frac{27^1}{3} = 9$

berechnet, da Potenzieren Vorrang vor Dividieren hat. In diesem Zusammenhang spricht man von „RECHENHIERARCHIE“, d. h. einer Vorrangsregelung, die angibt, in welcher Reihenfolge verschiedene Rechenschritte abgearbeitet werden. Diese Rechenhierarchie sieht in der Basic-Sprache so aus:

- zuerst werden in Klammern stehende Ausdrücke berechnet. Dabei wird mit der innersten Klammer begonnen.
- Bei den Operationen gilt die Rangfolge: Potenzierung, dann Multiplikation oder Division und zuletzt Addition oder Subtraktion.
- Gleichrangige Ausdrücke werden von links nach rechts abgearbeitet.

Man muß immer ebensoviele Klammern schließen, wie man geöffnet hat, sonst gibt der Computer eine Fehlermeldung aus. Bei der Umwandlung von komplizierten Formeln in das Basic-Format muß man sehr aufmerksam sein — eine falsch gesetzte oder fehlende

Klammer kann ein völlig falsches Ergebnis verschulden. Hier noch ein Beispiel für diese Umwandlung einer Formel: Die Formel

$$\frac{5 \cdot (3+4)^2}{2^2 \cdot (3+4)}$$

muß in Basic lauten:

```
>PRINT 5★ ( 3+4 ) [ 2   ( 2 [ 2★ ( 3+4 ) )
 8 . 75
READY
>■
```

Geben Sie nun folgenden Befehl ein:

```
>PRINT 999999+1
 1E+06
READY
>■
```

Was hat diese Antwort zu bedeuten?
Ganz einfach: Alle Zahlen, die größer als 999999 oder kleiner als 0.01 sind, werden in der sogenannten EXPONENTIALSCHREIBWEISE ausgegeben, d. h., sie werden in Mantisse (Anteil vor dem „E") und Exponent (Anteil nach dem „E") aufgespalten, damit die Zahlen in der Ausgabe nicht zu lang werden.

Zum Verständnis möchte ich 2 Beispiele bringen, die zeigen, wie man die Exponentenschreibweise aufzuschlüsseln hat:

$$3.5E+06 = 3.5 \cdot 10^6 = 3.5 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 3500000$$

$$3.5E-04 = 3.5 \cdot 10^{+4} = \frac{3.5}{10 \cdot 10 \cdot 10 \cdot 10} = 0.00035$$

Das Colour-Genie hat einen Rechenbereich von 1.70141E−38 bis 1.70141E+38, d. h., Sie können bis zu 39stellige Zahlen verarbeiten.

Noch einige wenige Hinweise zum Rechnen mit dem Genie: Normalerweise wird mit 6 Stellen Genauigkeit gerechnet. Es gibt jedoch auch die Möglichkeit, mit 16 Stellen Genauigkeit zu rechnen, doch davon mehr im nächsten Kapitel, das sich mit dem Thema Variablen beschäftigen wird.

Natürlich kann Ihr Genie auch mathematische Funktionen, wie Sinus oder Logarithmus berechnen — die nötigen Befehle lernen Sie noch kennen. Schon mal vorab ein Beispiel:

```
>PRINT LOG ( 5 )
 1.60944
READY
>■
```

1.60944 ist der natürliche Logarithmus von 5.

Da die Rechengenauigkeit des Genies beschränkt ist, kann es zu sogenannten Rundungsfehlern kommen. Ein Beispiel:

```
>PRINT 3 [ 4
 81.0001
READY
>■
```

Richtig wäre natürlich 81 und nicht 81.0001 — Sie sollten also immer auf der Hut vor solchen Rundungsfehlern sein und Ihr Ergebnis dahingehend prüfen.

4. Variablen

Variablen spielen bei der Programmierung in der Basic-Computersprache eine wichtige Rolle. Mit dem Wort „Variable“ wird eigentlich ein ganz bestimmter Speicherraum, der durch den Namen der Variablen gekennzeichnet ist, benannt. Dieser Speicherraum wird im Hauptspeicher des Computers in dem Augenblick reserviert, in dem dieser Variablen im Programm zum ersten Mal ein bestimmter Wert zugewiesen wird. Der Zuweisungsbefehl sieht allgemein so aus:

LET Variable = zuzuweisender Wert
(Der Befehl „LET“ kann auch weggelassen werden)

z. B.:

```
>LET A=5
READY
>■
```

Damit ist der Variablen A der Wert 5 zugewiesen.
Dies kann man folgendermaßen überprüfen:

```
>PRINT A
 5
READY
>■
```

Geben Sie nun ein

```
>PRINT B
 0
READY
>■
```

Der Variablen B ist noch kein Wert zugeordnet, deshalb wird sie mit 0 angegeben.

Man kann einer Variablen auch das Ergebnis einer Rechnung zuweisen. Ein Beispiel:

```
>B=A+10★15—1
READY
>PRINT B
 154
READY
>■
```

Eine Variable kann auch in der eigenen Zuweisung benutzt werden:

```
>C=10
READY
>C=C+1
READY
>PRINT C
 11
READY
>■
```

Der Befehl „C=C+1“ ist keine mathematisch richtige Gleichung. Daher spricht man ihn am besten so: C wird zu C plus eins.

Geben sie Folgendes ein:

```
>TEST=0.009
READY
>PRINT TEST
 9E—03
READY
>■
```

Wie Sie sehen, kann man Variablen auch längere Namen geben. Dabei gilt allgemein:

— Das erste Zeichen muß ein Buchstabe sein.
— Alle folgenden Zeichen können wahlweise ein Buchstabe oder eine Zahl sein, alle anderen Zeichen sind unzulässig.
— Zur Unterscheidung dienen nur die ersten zwei Zeichen, d. h. der Computer kann z. B. die Variablen „TEST" und „TEXT" nicht unterscheiden!
— In einem Variablennamen dürfen keine Basic-Schlüsselwörter, also keine Befehlswörter, vorkommen.
  Der Variablenname „XPRINTY" ist z. B. unzulässig, weil er den Befehl „PRINT" enthält.
  (Siehe Liste der BASIC-Befehlswörter.)

Es gibt zwei grundsätzlich verschiedene Arten von Variablen:

1. ZAHLEN-Variablen, auch numerische Variablen genannt.
   Wenn der Computer eine Zahlen-Variable findet, entnimmt er deren Zahlenwert aus dem Speicher und kann dann diesen Wert in einem Rechenvorgang verarbeiten.
   Bei den Zahlenvariablen gibt es 3 verschiedene Typen:
   — Einfache Genauigkeit (6stellig)
   — Doppelte Genauigkeit (16stellig)
   — Ganzzahl-Variablen (INTEGER-Variablen)
   Bis jetzt haben wir nur Variablen mit einfacher Genauigkeit benutzt. Die anderen Typen werden Sie gleich kennenlernen.

2. ZEICHENKETTEN-Variablen, auch String-Variablen genannt, zur Abspeicherung beliebiger Texte bis zu 255 Zeichen Länge. Mit diesen Variablen wollen wir uns nach den verschiedenen Zahlenvariablen beschäftigen.

Nun zu den 3 verschiedenen Typen von Zahlenvariablen:

Variablen mit einfacher Genauigkeit

Alle Variablen, die wir bis jetzt benutzt haben, waren Variablen mit einfacher (= 6stelliger) Genauigkeit.

Normalerweise sind alle Variablen, die hinter ihrem Namen kein Sonderzeichen haben, Variablen mit einfacher Genauigkeit. Wie Sie jedoch später sehen werden, gibt es die Möglichkeit, Variablen per Befehl auf eine andere Genauigkeit zu definieren. Wenn man in diesem Falle noch mit einfacher Genauigkeit rechnen will, muß dies durch ein ! hinter dem Variablennamen angegeben werden.

Ist dies aber nicht der Fall, so sind Variablen mit und ohne ! identisch. Ein Beispiel:

```
>A !=3.456
READY
>PRINT A
 3.456
READY
>■
```

Variablen mit doppelter Genauigkeit

Variablen mit doppelter Genauigkeit rechnen auf 16 Stellen genau. Sie werden durch ein # hinter dem Variablennamen gekennzeichnet. Geben Sie Folgendes ein:

```
>A# =1 / 3
READY
>PRINT A#
 .3333333432674408
READY
>■
```

Das Ergebnis ist zwar 16stellig, nach der 7. Stelle aber falsch!

Das liegt an einer Eigenart des Genie-Basics: Wenn ein Ergebnis mit 16stelliger Genauigkeit erreicht werden soll, muß mindestens ein Teil der Zuweisung ebenfalls als 16-stellig-genau definiert sein.
Man kann nun auch eine Zahl als 16-stellig-genau definieren, indem man ein # anfügt.

So bekommen wir das gewünschte Ergebnis der obigen Rechnung:

```
>A# =1# / 3 #
READY
>PRINT A#
 .3333333333333333
READY
>■
```

Dasselbe geht auch ohne vorherige Variablenzuweisung:

```
>PRINT 1# / 3#
 . 3333333333333333
READY
>■
```

Der Nachteil von Variablen mit doppelter Genauigkeit ist, daß sie mehr Speicher brauchen und relativ langsam berechnet werden.

Noch ein Hinweis: Bei Exponentialdarstellung wird bei Variablen mit doppelter Genauigkeit ein „D“ statt des „E“ bei Variablen mit einfacher Genauigkeit ausgegeben.

Integer-Variablen

Integer-Variablen, auch Ganzzahlvariablen genannt, werden durch ein %-Zeichen hinter dem Namen gekennzeichnet. Der Bereich von Integer-Variablen reicht von —32768 bis +32767.

Nachkommastellen sind nicht möglich. Dafür haben Integer-Variablen den Vorteil, daß sie weniger Speicher brauchen und vom Computer erheblich schneller als die anderen beiden Zahlenvariablentypen bearbeitet werden.

Ein Beispiel für Integer-Variablen:

```
>A%=3.9
READY
>PRINT A%
 3
READY
>■
```

Sie sehen, daß der Nachkommaanteil einfach weggelassen wird, d. h., es wird nicht gerundet!

Hier noch einmal die 3 verschiedenen numerischen Variable auf einen Blick:

A	6 Stellen
A!	6 Stellen
A#	16 Stellen
A%	Ganzzahl

Dabei sind die Variablen A, A#, A% 3 Variablen, die völlig unabhängig voneinander sind:

```
>A=15.6
READY
>A#=2.34567890123
READY
>A%=200
READY
>PRINT A
 15.6
READY
>PRINT A#
 2.34567890123
READY
>PRINT A%
 200
READY
>■
```

Zeichenketten-Variablen

Zeichenketten-Variablen, auch String-Variablen genannt, dienen dazu, Texte zu bearbeiten. Rechnen kann man mit ihnen also nicht.

Eine Variable wird als Zeichenketten-Variable festgelegt, indem man direkt hinter den Namen ein Dollar-Zeichen setzt.

Ein Beispiel:

```
>A$="Text"
READY
>PRINT A$
Text
READY
>■
```

Wie Sie sehen, wird der Text bei der Zuweisung in Anführungszeichen gesetzt. Deshalb darf im Text selbst auch kein Anführungszeichen vorkommen!

Die maximale Länge des Textes, der in einer Zeichenketten-Variablen gespeichert werden kann, beträgt 255 Zeichen. Jedoch reserviert der Computer beim Einschalten nur Speicherplatz für 50 Zeichen.

Es gibt allerdings die Möglichkeit, diesen Speicherplatz individuell für das eigene Programm neu zu definieren.

5. Programmierung

Bis jetzt haben Sie alle Befehle so eingegeben, daß sie vom Computer direkt ausgeführt wurden. Diese Betriebsart wird man immer dann wählen, wenn man schnell irgendeine Rechnung ausführen will. Die weitaus wichtigeren Betriebsarten sind aber die Programmeingabe und der Programmablauf.

Im Prinzip besteht ein Basic-Programm aus einer oder mehreren Programmzeilen, von denen jede einen oder mehrere Befehle enthält. Diese werden dann vom Computer der Reihe nach automatisch abgearbeitet.

Dazu ein Beispiel:

```
>10 A=10
>20 A$="TEXT"
>30 PRINT A$
>40 PRINT A
```

Das Programm ist damit eingegeben.

Geben Sie nun folgenden Befehl ein:

```
>LIST
10 A=10
20 A$="TEXT"
30 PRINT A$
40 PRINT A
READY
>■
```

Wie Sie sehen, listet der „LIST"-Befehl das eingegebene Programm auf.

Gestartet wird ein Programm mit dem „RUN"-Befehl:

```
>RUN
TEXT
 10
READY
>■
```

Der Computer hat damit alle 4 Befehle hintereinander ausgeführt. In Zeile 10 hat er A den Wert 10 zugewiesen und in Zeile 20 A$ die Zeichenkette „TEXT". Zeile 30 gibt dann den Text von A$ aus und Zeile 40 den Wert von A.

Wenn Sie bei der Programmeingabe eine Zeile falsch eingegeben haben, geben Sie die entsprechende Zeile mit der gleichen Zeilennummer einfach neu ein. Die alte Zeile wird dabei automatisch gelöscht.

Das vorangegangene Beispielprogramm hat die Zeilennummern 10, 20, 30 und 40.

Im Genie-Basic kann man Zeilennummern von 0 bis 65529 benutzen. Bei der Wahl von Zeilennummern sollte man auf jeden Fall zwischen den einzelnen Zeilennummern einen gewissen Abstand lassen, im obigen Programm ist z. B. ein Abstand von 10 zwischen zwei Zeilennummern.

Diese Programmierweise hat folgenden Vorteil: Wenn zwischen zwei Zeilennummern noch Platz ist, kann man dort jederzeit neue Zeilen einfügen.

Das können Sie mit dem Programm ausprobieren, das Sie zuletzt eingegeben haben.

Angenommen Sie wollen noch zwei Zeilen einfügen, nämlich Zeile 15, die der Variablen B1 den Wert 2.56 zuweist, und Zeile 25, die diesen dann ausgibt. Dazu geben Sie Folgendes ein:

```
>15 B1=2.56
>25 PRINT B1
>LIST
10 A A=10
15 B1=2.56
20 A$="TEXT"
25 PRINT B1
30 PRINT A$
40 PRINT A
READY
>■
```

Mit dem „LIST“-Befehl sieht man, daß beide Zeilen eingefügt worden sind. Wenn Sie das Programm starten, werden alle 3 Werte ausgegeben:

```
>RUN
 2.56
TEXT
 10
READY
>■
```

Wenn Sie eine Zeile löschen wollen, geben Sie nur die Zeilennummer ein. Wenn Sie z. B. die Zeile 40 löschen wollen, geben Sie einfach eine 40 ein:

```
>40
READY
>LIST
10 A=10
15 B1=2.56
20 A$="TEXT"
25 PRINT B1
30 PRINT A$
READY
>■
```

Der „LIST“-Befehl zeigt, daß die Zeile 40 entfernt wurde.

Um das ganze Programm zu löschen, gibt man den „NEW“-Befehl ein:

```
>NEW
(Der Bildschirm wird gelöscht)
READY
>LIST
READY
>■
```

Dabei werden sowohl das Programm als auch alle Variablen gelöscht.

Wie schon erwähnt, lassen sich auch mehrere Befehle in eine Zeile bringen. Dies geschieht einfach dadurch, daß man die einzelnen Befehle durch einen Doppelpunkt trennt. Der Vorteil der Aneinanderreihung von Befehlen in einer Zeile liegt darin, daß Speicherplatz eingespart wird, da der Computer weniger Zeilennummern speichern muß.

Ein Beispiel: (Geben Sie vor der Programmeingabe den Befehl „NEW“ ein, um ein eventuell noch vorhandenes Programm zu löschen.)

```
>10 A=10 : B1=2.56 : A$="TEXT" : PRINT B1 : PRINT A$ : PRINT A
>RUN
 2.56
TEXT
 10
READY
>■
```

Sie sehen, daß dieses Programm genauso läuft wie das Programm, bei dem Sie alle Befehle in einzelne Zeilen geschrieben haben. Wie Sie später noch genau sehen werden, gibt es jedoch bei der Aneinanderreihung von Befehlen eine Einschränkung, und zwar die, daß man mit dem „GOTO“-Befehl eine Programmzeile von einer anderen aus anspringen kann. Es ist aber nicht möglich, einen Befehl mitten in einer Programmzeile anzuspringen, d. h., man muß einen Befehl, der angesprungen werden soll, in eine neue Basic-Programmzeile zu schreiben.

Noch ein Wort zu den Leerzeichen in Programmzeilen:
Prinzipiell sind zwischen Befehlen, Variablen etc. keine Leerzeichen erforderlich.

Folgende Zeile führt also genau die Befehle aus, wie die obige Zeile mit Leerzeichen:

```
>10A=10:B1=2.56:A$="TEXT":PRINTB1:PRINTA$:PRINTA
```

Ein Programm bleibt jedoch übersichtlicher, wenn man Leerzeichen einfügt. Der Nachteil ist, daß auch Leerzeichen Speicherplatz belegen. Deshalb wird man sie in langen Programmen eher weglassen, um Speicher zu sparen. In Texten darf man die Leerzeichen selbstverständlich nicht weglassen!!

Wenn Sie dieses Handbuch bis hierin aufmerksam gelesen haben, sind Sie nun mit den Grundprinzipien der Basic-Programmierung recht gut vertraut.

Was Ihnen also jetzt noch fehlt, ist einfach die Kenntnis neuer Befehle. Um Ihnen diese zu vermitteln, sind im Folgenden auch Teile von Dr. Hans-Joachim Sachts Buch „Genie-Basic leicht gelernt“ sowie Teile der Anleitung für das Genie I/II übernommen. Diese Bücher wurden jedoch für das Genie I/II geschrieben, das eine andere Tastatur und keine Möglichkeit der Farbgestaltung auf einem Farbfernseher oder Farbmonitor hat. Daher werden wir uns, bevor neue Programmbefehle kommen, noch kurz mit der Tastatur und mit den Farben des Colour-Genies beschäftigen.

Außerdem wird vorher noch die Benutzung des Kassettenrekorders erklärt.

6. Die Tastatur

Die Tastatur ist ein wichtiger Bestandteil Ihres Computers — sozusagen die Mensch-Maschine-Schnittstelle. Zusätzlich zur Tastatur bietet das Colour-Genie die Möglichkeit, zwei Steuerknüppel (analog joysticks), zwei externe Tastenfelder (external keypads) und einen Lichtgriffel (lightpen) anzuschließen. Diese Eingabemittel können Sie als Zusatzgeräte erwerben. In jedem Falle werden Sie aber die meisten Eingaben über die Tastatur vornehmen.

Die Tastatur des Colour-Genie ähnelt zwar einer Schreibmaschinentastatur, hat aber viele Sonderfunktionen. Deshalb sollten Sie sich mit den Funktionen der einzelnen Tasten genau vertraut machen.

Man kann 4 Tastengruppen unterscheiden, die jeweils unterschiedliche Funktionen haben:

- Buchstabenblock der Haupttastatur,
- Ziffernblock der Haupttastatur,
- Steuertasten und
- Programmierbare Funktionstasten.

Gehen wir zuerst die *Steuertasten* durch:

RST

Wenn beide Reset-Tasten gleichzeitig niedergedrückt werden, wird der Computer gezwungen, in den Eingabemodus zurückzukehren. Programm und Variablen werden dabei nicht gelöscht. (Sog. „Warmstart") Die Reset-Tasten braucht man immer dann, wenn der Computer sich in einer Endlosschleife befindet, die unterbrochen werden soll (z. B. bei fehlerhaftem Einlesen einer Kassette).

Wenn Sie die beiden <RST>-Tasten *und* die <R>-Taste zusammen drücken, geht der Computer zur „MEM-SIZE? ■"-Frage, genau so, als ob Sie den Computer neu eingeschaltet hätten.
Programm und Variablen werden gelöscht. (Sog. „Kaltstart")

CLEAR

Diese Taste löscht den Bildschirm und der Cursor springt in die linke obere Bildschirmecke.

BREAK

Die <BREAK>-Taste unterbricht ein Programm, die Zeile in der der Abbruch erfolgte wird angegeben. Z. B.:

```
>10 PRINT "HALLO" : GOTO 10
>RUN
HALLO
HALLO
HALLO
u.s.w.    <BREAK>
Break in 10
READY
>■
```

Auch der „LIST"-Befehl kann mit <BREAK> abgebrochen werden.

Die zweite Funktion der <BREAK>-Taste besteht darin, vom Feingrafik-Modus zum Text-Modus zurückzukommen.

RPT RPT bedeutet Repeat, zu deutsch Wiederholung. Durch Drücken der <RPT>-Taste wird das zuletzt eingegebene Zeichen solange wiederholt, bis Sie die <RPT>-Taste wieder loslassen.

SHIFT Durch gleichzeitiges Drücken der <SHIFT>- und der <@>-Taste wird ein Programm (oder ein Listing) „eingefroren", d. h., die Ausführung angehalten, ohne daß sich auf dem Bildschirm etwas verändert. Durch Drücken irgendeiner anderen Taste wird die Ausführung wieder aufgenommen.

← Diese Taste löscht das zuletzt eingegebene Zeichen.

SHIFT ← <SHIFT> und <←> zusammen gedrückt löscht die komplette Eingabe.

→ Tabulator-Funktion: Der Cursor springt auf die nächste Tabulatorposition auf dem Bildschirm (jedes 8. Zeichen).

CTRL Control-Taste. Drücken Sie die <CTRL>-Taste und anschließend eine der Ziffertasten von <1>...<8>. Sie sehen, daß alle folgenden Zeichen in der Farbe dargestellt werden, die auf der Vorderseite der Ziffertasten aufgedruckt ist.

<CTRL> und dann			
<1>	WHT	white	weiss
<2>	GRN	green	grün
<3>	RED	red	rot
<4>	YEL	yellow	gelb
<5>	ORG	orange	orange
<6>	BLU	blue	blau
<7>	CYN	cyan	cyan (blaugrün)
<8>	MAG	magenta	magenta (rotviolett)

CTRL MOD SEL Wenn Sie <CTRL> gleichzeitig mit <MOD SEL> drücken, schaltet der Computer in den hochauflösenden Grafik-Modus. Als Beispiel geben Sie folgende 2 Befehle ein:

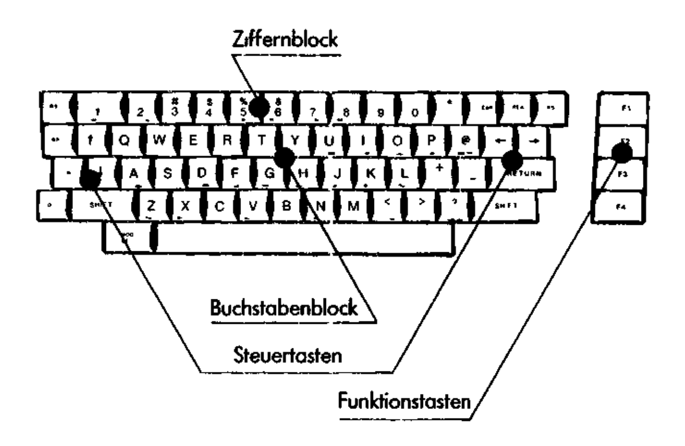
>FCOLOUR 3 : CIRCLE 80, 48, 40

Dieser Befehl zeichnet einen orangefarbigen Kreis.

Nun drücken Sie <CTRL> und <MOD SEL> und der Kreis wird sichtbar! (Je nach Symmetrierung Ihres Fernsehers kann der Kreis aber auch als Elipse dargestellt werden.)

Wenn Sie dann <BREAK> drücken, kehren Sie in den Text-Modus zurück.

RETURN Bewirkt die Übernahme und Ausführung einer Eingabe.



MOD SEL Auch diese Taste hat mehrere Funktionen: Wenn Sie die <MOD SEL>-Taste und anschließend irgendeine Buchstabentaste drücken, sehen Sie, daß nicht ein Buchstabe ausgegeben wird sondern das Grafikzeichen, das links auf der Vorderseite der jeweiligen Buchstabentaste aufgedruckt ist. Wollen Sie das rechts aufgedruckte Zeichen haben, drücken Sie <SHIFT> und dann die entsprechende Buchstabentaste.

Um wieder normale Buchstaben eingeben zu können, drücken sie einfach nochmals die <MOD SEL>-Taste.

Wie schon erklärt, schaltete <CTRL> <MOD SEL> in den Grafik-Modus um.

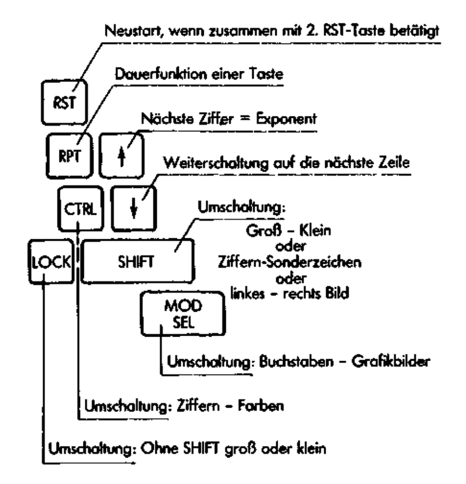
Die <MOD SEL>-Taste hat aber noch eine dritte Funktion: Wenn man beim Einschalten des Computers die <MOD SEL>-Taste niedergedrückt hält, werden die 4096 Bytes Speicherplatz, die sonst für die hochauflösende Grafik reserviert sind, dem verfügbaren Programmspeicher beigefügt. In diesem Fall stehen insgesamt 14012 Bytes zur Verfügung(1 Byte entspricht 8 Bits, in einem Byte kann z. B. ein Buchstabe oder ein Befehlswort gespeichert werden). Man darf dann aber auf keinen Fall irgendwelche Befehle für die hochauflösende Grafik ausführen!!!

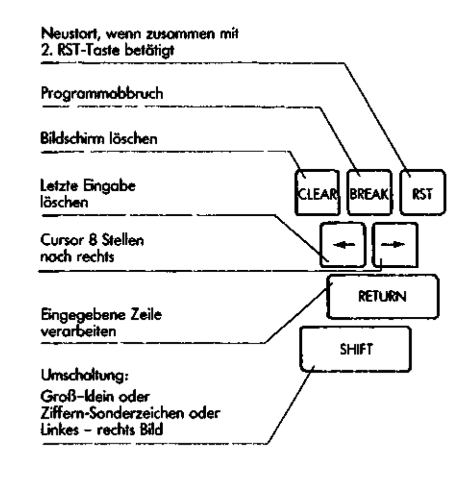
Wenn die <MOD SEL>-Taste beim Einschalten nicht niedergedrückt wird, stehen 9916 Bytes für ein Basicprogramm zur Verfügung.

SHIFT Zusammen mit anderen Steuertasten hat die <SHIFT>-Taste diverse Steuerfunktionen — dies ist bei den vorigen Tastenerklärungen bereits gezeigt worden. Ansonsten bringt die <SHIFT>-Taste, zusammen mit einer Buchstabentaste gedrückt, Kleinschrift, und, zusammen mit einer Zifferntaste gedrückt, ein Sonderzeichen.

LOCK Dies ist genaugenommen keine Taste sondern ein Schalter. Wenn der <LOCK>-Schalter niedergedrückt ist, hat das dieselbe Funktion, als wenn man die <SHIFT>-Taste dauernd gedrückt halten würde.
Durch nochmaliges Drücken rastet dann der <LOCK> -Schalter wieder aus.

[↓] Bewegt den Cursor in die nächste Zeile (Linefeed). Linefeeds dürfen in Befehlen nicht verwendet werden. Sinnvoll ist ihre Anwendung aber in Zeichenketten immer dann, wenn man eine Leerzeile ausgeben will.





Die programmierbaren Funktionstasten

Rechts neben dem Haupttastaturblock befinden sich 4 Tasten, die mit F1, F2, F3 und F4 beschriftet sind. Dies sind die 4 programmierbaren Funktionstasten des Colour-Genie.

Solange Sie diese Tasten nicht umprogrammiert haben, sind sie mit folgenden 8 Befehlen belegt:

ohne <SHIFT>		mit <SHIFT>
<F1>	LIST	RENUM
<F2>	RUN	DELETE
<F3>	AUTO	CLOAD
<F4>	EDIT	CSAVE

Wenn Sie z. B. <F2> drücken, erscheint der Befehl „RUN“. Drücken Sie <SHIFT> <F3> erscheint der Befehl „CLOAD“.

Wie kann man nun einen neuen Befehl auf eine Funktionstaste legen?

Das allgemeine Format dazu ist:

FKEY n = "Text"

mit n=1...8 und einer Textlänge von maximal 7 Zeichen.

Mal angenommen, Sie wollen den Befehl „GOTO“ auf <F1> legen, dann sieht der Befehl zur Umprogrammierung der <F1>-Taste so aus:

```
>FKEY1="GOTO"
READY
>■
```

Drücken Sie <F1> und es erscheint „GOTO“.

Wenn Sie beim "FKEY"-Befehl die hinteren Anführungszeichen weglassen, wird der definierte Befehl direkt ausgeführt. Dies kann zum Beispiel beim „RUN“ und „SYSTEM“-Befehl sinnvoll sein, weil danach ohnehin meist ein <RETURN> folgt.

Zu den Buchstaben- und Zifferntasten läßt sich eigentlich nichts weiter sagen, denn diese Tasten funktionieren genau wie bei einer Schreibmaschine, außer daß <Y> und <Z> im Vergleich zur deutschen Tastatur vertauscht sind, und daß normalerweise groß und mit <SHIFT> klein geschrieben wird.

Tastencode bei INKEY $ ohne/mit <SHIFT>	
92/124	F1
93/125	F2
94/126	F3
95/127	F4

7. Anschluß eines Kassettenrekorders

Beim Ausschalten verliert des Colour-Genie alle Programme und Daten. Daher wird man Programme, die man später noch braucht, auf einer Kassette sichern. Das Colour-Genie schreibt und lädt mit einer recht hohen Aufzeichnungsrate von 1200 Baud, dies entspricht ca. 150 Zeichen pro Sekunde.

Der Kassettenrekorder (nicht im Lieferumfang!) wird mit dem (mitgelieferten) Kabel an die DIN-Buchse hinten rechts auf der Rückseite des Computers angeschlossen. Als Rekorder sollten Sie einen handelsüblichen Rekorder benutzen, der die Buchsen (für 3.5 mm Klinkenstecker) EAR (Kopfhörer) und MIC (Mikrofon) hat.

Dabei reicht ein einfaches Monogerät ohne automatische Aussteuerung — Rauschunterdrückungsschaltungen u. ä. sind auf jeden Fall störend.

Ein Bandzählwerk ist beim Suchen von Programmen auf Kassette eine große Hilfe.

Das weiße Kabel stecken Sie in die EAR-Buchse, das schwarze Kabel in die MIC-Buchse.

Als Kassetten sind gute (= Drop-out freie) Eisenoxid-Bänder am besten geeignet; Chrom- oder gar Reineisenbänder bringen keine Vorteile.

Das Programm K abspeichern

[C] [S] [A] [V] [E] ["] [K] ["]

[REC] [PLAY]

[RETURN]

Das Programm K laden

[C] [L] [O] [A] [D] ["] [K] ["]

[RETURN]

[PLAY]

8. CSAVE, VERIFY, CLOAD

Dies sind die 3 Befehle, mit denen Sie ein Basic-Programm auf die Kassette schreiben, eine Aufnahme verifizieren und von der Kassette laden können.

CSAVE

Dieser Befehl schreibt ein Programm auf die Kassette. Benutzen Sie ihn auf folgende Weise:

— Spulen Sie das Band an die Stelle, an der die Aufzeichnung beginnen soll. Am Anfang einer Kassette achten Sie darauf, daß der nicht beschreibbare Vorspann vorüber ist.

— Geben Sie

  >CSAVE "N"

  ein, wobei N für irgendeinen Buchstaben steht, mit dem Sie der Aufzeichnung einen Namen geben.

— Drücken sie gleichzeitig die PLAY- u. RECORD-Tasten Ihres Rekorders.

— Drücken Sie die <RETURN>-Taste. Das Programm wird nun aufgezeichnet. Von wichtigen Programmen sollten Sie aus Sicherheitsgründen mehr als eine Aufnahme machen.
  (Sicherungskopie!!)

VERIFY

Dieser Befehl vergleicht ein Programm, das sich im Speicher befindet, mit einer Bandaufzeichnung. Man benutzt diesen Befehl dazu, ein mit „CSAVE" aufgezeichnetes Band zu testen. Gehen Sie dabei folgendermaßen vor:

— Spulen sie das Band zum Anfang der Aufnahme.

— Geben Sie

  >VERIFY

  ein und drücken Sie die <RETURN>-Taste.

— Drücken Sie die PLAY-Taste Ihres Rekorders.

Wenn die Aufzeichnung gefunden ist, erscheinen in der rechten oberen Bildschirmecke zwei Sternchen, von denen das rechte zur Kontrolle des Ladevorgangs blinkt.

Erfolgreiche Verifizierung endet mit der „READY"-Meldung. Sollte ein Fehler auftreten, meldet sich der Computer mit „BAD". In diesem Fall probieren Sie eine andere Lautstärkeeinstellung und verifizieren die Aufnahme erneut. Sollte dies nicht helfen, nehmen Sie eine andere Kassette.

CLOAD

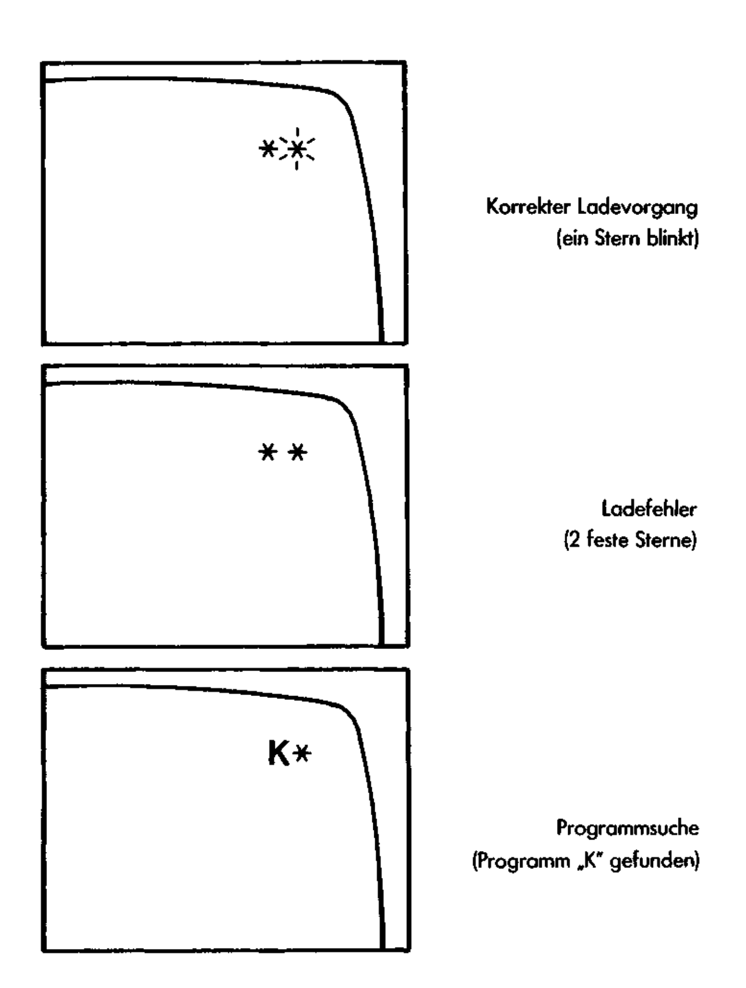
Mit diesem Befehl wird ein Basicprogramm von der Kassette in den Speicher geladen, wobei ein vorher evtl. im Speicher stehendes Programm gelöscht wird. Gehen Sie folgendermaßen vor:

— Spulen Sie das Band an den Anfang der Aufzeichnung.

— Geben Sie

>CLOAD „N“

ein, wobei N für den Kennzeichnungsbuchstaben steht. Wenn Sie nur CLOAD eingeben, beachtet der Computer die Kennzeichnung nicht und lädt das Basicprogramm, welches er als nächstes auf dem Band vorfindet.
Wird ein Programm mit einem anderen Kennzeichnungsbuchstaben gefunden, wird dieser anstelle des linken Sternchens angezeigt, das falsche Programm wird nicht geladen.
Wenn das richtige Programm gefunden ist, erscheinen 2 Sternchen in der rechten oberen Bildschirmecke, wobei das rechte zur Kontrolle blinkt. Sollte dieses Sternchen nicht mehr blinken und der Computer nicht zu „READY“ kommen, liegt ein Ladefehler vor. Drücken Sie dann beide <RST>-Tasten und versuchen Sie das Programm mit einer anderen Lautstärke zu laden.
Ein Ladefehler liegt auch dann vor, wenn im Programm unverständliche oder falsche Zeilen zu finden sind.



Korrekter Ladevorgang
(ein Stern blinkt)

Ladefehler
(2 feste Sterne)

Programmsuche
(Programm „K“ gefunden)

9. Der COLOUR-Befehl

Im 7. Kapitel haben sie bereits gelernt, daß man über die Tastatur mit <CTRL> + <1>...<8> die Farbe verändern kann. Diese Änderung wirkt sich aber nur während der Eingabe aus, in einem laufenden Programm kann man mit dieser Tastenkombination die Farbe nicht verändern.

Mit dem „COLOUR"-Befehl kann man nun innerhalb eines Programms die Farbe wechseln, wobei man zwischen 16 Farben wählen kann.

Man kann den „COLOUR"-Befehl auch direkt eingeben:

```
>COLOUR 15
READY
>■
```

„READY" erscheint nun in lila.

In einem Programm sähe die Verwendung z. B. so aus:

```
>10 COLOUR 2 : PRINT"Ausgabe in grün"
>20 COLOUR 4 : PRINT"Ausgabe in orange"
>RUN
Ausgabe in grün
Ausgabe in orange
READY
>■
```

Folgendes Programm zeigt Ihnen alle 16 Farben des Colour-Genies:

```
>10 FOR A = 1 TO 16
>20 COLOUR A
>30 PRINT "Das ist Farbe Nr."; A
>40 NEXT A
```

Mit dem COLOUR-Befehl kann man beliebig oft die Farbe der ausgegebenen Zeichen ändern, und so ansprechende Farbbilder auf dem Fernsehschirm erzeugen.

Auch zum Programmieren sollten Sie sich eine Farbe aussuchen, die Ihnen gefällt und auf Ihrem Fernseher gut zu lesen ist.

Die folgenden Kapitel stammen, wie bereits erwähnt, aus einem Handbuch, das für Genie I/II geschrieben wurde — da das Genie I/II keine Möglichkeit der Farbwidergabe bietet, ist in den kommenden Programmbeispielen der Colour-Befehl nicht zu finden. Fügen Sie ihn aber nach Belieben ein, wo es Ihnen sinnvoll erscheint. — Ohnehin sind Sie jetzt langsam so weit, daß Sie nicht nur die gegebenen Programmbeispiele ausprobieren, sondern nun auch eigene Programme entwickeln und testen sollten!

10. Der PRINT-Befehl

Der PRINT-Befehl ist der Ausgabebefehl für den Computer. Er weist den Computer an, das, was hinter dem Wort PRINT im Befehl steht, als Anzeige auf dem Bildschirm auszugeben.

Dabei gibt es grundsätzlich zwei Möglichkeiten. Folgt nach dem PRINT-Befehl ein Anführungszeichen, so bedeutet dies, daß anschließend eine Zeichenkette kommt. In diesem Fall wird also die komplette Zeichenkette bis zu den abschließenden Anführungsstrichen zur Anzeige gebracht.

Steht nach dem PRINT-Befehl kein Anführungszeichen, betrachtet der Computer alles, was dann folgt, als Rechenausdruck. Er berechnet das dazugehörige Ergebnis und bringt dies zur Anzeige.

Steht nach dem PRINT-Befehl eine Variable, wird der unter ihrem Namen abgespeicherte Wert ausgegeben oder in die Rechnung mit einbezogen.

Eine besondere Rolle nach einem PRINT-Befehl spielen die Satzzeichen Semikolon (;) und Komma (,), denn mit ihrer Hilfe können mehrere Ausgaben durch einen Befehl veranlaßt werden.

— wird ein Semikolon zur Trennung verwendet, so wird dicht hintereinander ausgegeben,
— bei Trennung durch ein Komma wird in Kolonnen mit jeweils 10 Zeichen Abstand angezeigt.

Der PRINT-Befehl ohne nachfolgendes Argument erzeugt eine Leerzeile. Da der Doppelpunkt (:) verwendet werden kann, um mehrere Befehle in eine Zeile zu schreiben, ergeben mehrere PRINT-Befehle durch Doppelpunkte getrennt mehrere Leerzeilen.

Der Zeilentransport nach einem PRINT-Befehl kann durch ein Semikolon am Ende der Zeile unterdrückt werden.

```
>10 A = 25 : B = 10
>20 PRINT A
>30 PRINT A ★ 5
>40 PRINT A / B
>50 PRINT"A / B"
>60 PRINT"A / B
>RUN
  25
  125
  2.5
A / B
A / B
READY
>■
```

Die Möglichkeiten des PRINT-Befehls

```
>10 A = 5 : B = 8
>20 PRINT A ; B
>30 PRINT
>40 PRINT A , B
>50 PRINT A ;
>60 PRINT B
>RUN
 5  8

 5          8
 5  8
READY
>■
```

Bedeutung der Satzzeichen

Nun müssen sie noch einige Besonderheiten bei den Trennungszeichen innerhalb von PRINT-Befehlen beachten.

Bei der Ausgabe von positiven Zahlen wird vor die Zahl eine Leerstelle gesetzt, in die bei einer negativen Zahl das Minuszeichen kommt. Auch nach jeder Zahl erfolgt die Ausgabe einer Leerstelle.

Textstellen, die als Zeichenketten behandelt werden sollen, müssen in Anführungsstrichen stehen. Alle Leerzeichen innerhalb der Anführungszeichen werden auch als Leerzeichen ausgegeben. Vom Computer werden vor und hinter Zeichenketten keine Leerzeichen gesetzt.

Ist die in einer Kolonne auszugebende Information länger als 10 Zeichen, wird in die nächste Kolonne hineingeschrieben. Die nächste Ausgabe erfolgt dann beim nächstfolgenden Kolonnenanfang.

Wenn Sie eine Ausgabe in Kolonnen programmieren, sollten Sie sicher sein, daß niemals mehr als 10 Zeichen auszugeben sind.

Sie sollten auch niemals versuchen, eine Ausgabe mit mehr als 4 Kolonnen zu programmieren. Ihr Genie nimmt zwar auch mehr Kolonnen zur Ausgabe an, aber dies hat nur Sinn beim Anschluß eines breiten Druckers. Auf dem Bildschirm wird dann die 5. Kolonne unter der 1. gedruckt, da die Anzeige selbsttätig nach der 40. Spalte in der Spalte 1 der neuen Zeile erfolgt.

```
>LIST
10 A = 10 : B = —12
20 A' = "KETTE" : B$ = "MIT"
30 PRINT A ; A ; B ; B ; A ; B
40 PRINT A$ ; A$ ; A$ ; B$ ; A$
READY
>RUN
 10  10  —12  —12  10  —12
KETTEKETTEKETTE MIT KETTE
```

Druckbild bei Engdruck

```
>10 A$ = "123456789012345"
>20 PRINT A$ ,' a$
>RUN
123456789012345 123456789012345
READY
>■
```

Druckbild bei Kolonnendruck, maximal 15 Zeichen in einer Kolonne

Sie können aber auch wie mit dem Tabulator der Schreibmaschine die Anfänge der Tabellenspalten selbst bestimmen. Hierzu dient die TAB(.)-Funktion. Wenn Sie in die Klammer eine Zahl zwischen 0 und 39 setzen und diese Funktion vor den zu druckenden Ausdruck schreiben, beginnt die Ausgabe erst in der durch die Zahl genannten Spalte+1. Es ist darauf zu achten, daß zwischen TAB und der Klammer kein Zwischenraum stehen darf.

Bei Ihrem Genie gibt es dann noch einen weiteren sehr nützlichen Befehl, mit dem die Bildschirmausgabe an einer beliebigen Stelle programmiert werden kann. Der Bildschirm des Genies zeigt 24 Zeilen mit je 40 Zeichen. Es gibt also insgesamt 960 Anzeigepositionen, die in der linken oberen Ecke mit 0 beginnend bis 959 in der rechten unteren Ecke durchnumeriert sind. Mit dem Befehl PRINT @ und einer nachfolgenden Ziffer können Sie die Ausgabe in einer (Beispiel!) beliebigen Bildschirmposition beginnen lassen.

Als Besonderheit kennt Ihr Genie dann noch den PRINTUSING-Befehl, mit dem die Dezimalzahlen immer mit den Punkten für die Dezimaltrennung schön untereinander ausgegeben werden können. Für die Benutzung dieses Befehls ist es zunächst notwendig, das Format der Ausgabe als Zeichenkette zu definieren. Für jede anzuzeigende oder zu druckende Stelle setzt man ein Nummernzeichen (#) und fügt den Dezimalpunkt dort ein, wo er hingehört. Haben die auszugebenden Zahlen mehr Nachkommastellen als gewünscht, erfolgt eine korrekte Rundung der letzten angezeigten Zahl.

Es gibt noch eine ganze Menge weiterer Möglichkeiten, diesen Befehl für das Schreiben von Dollarbeträgen zu verwenden, die aber für deutsche Verhältnisse nicht wichtig sind. Beachten müssen Sie aber das folgende:

— Der Befehl muß genauso geschrieben werden wie in den Beispielen gezeigt; mit einem Semikolon vor der Zahlen-Variablen.
— Hat die Zahl mehr Vorkommastellen als formatiert, wird vor die Zahl das Prozentzeichen (%) gesetzt. Die Rundung kann dann falsch sein.
— Mehrere formatierte Ausdrucke in einer Zeile sind nach den im Beispiel gezeigten Methoden möglich.

```
>10 A = 99
>20 PRINT TAB( 5 ) A ; TAB( 15 ) A
>RUN
      99         99
READY
>■
```

Tabellierung mit TAB-Funktion

```
READY
>10 A = 123.456
>20 B = 5432.1234
>30 U$ = "###.##"
>40 PRINT USING U$ ; A ;
>50 PRINT USING U$ ; B ;
>RUN
123.46
%5432.12
READY
>■
```

Formulierung mit PRINT USING

```
>10 A = 123.456
>20 B = 432.1267
>30 U$ = "###.##      "
>40 PRINT USING U$ ; A
>50 PRINT TAB( 10 ) USING U$ ; B
>60 PRINT USING U$ ; A ; B
>RUN
123.46      432.13
123.46      432.13
READY
>■
```

Mehrere Formatierungen in einer Zeile

11. Der INPUT-Befehl

Im Gegensatz zu vielen anderen höheren Programmiersprachen gibt es in BASIC auch einen Befehl, mit dem während des Programmlaufs Daten durch den Benutzer eingegeben werden können. Verwendet wird hierfür das Befehlswort INPUT. Jedesmal wenn der INPUT-Befehl im Programm erscheint, unterbricht der Computer den Programmablauf und wartet, bis über die Tastatur eine entsprechende Eingabe erfolgt ist. Zum Zeichen für den Bedienenden, daß der Computer sich in Wartestellung befindet, erscheint auf dem Bildschirm ein Fragezeichen.

Mit dem INPUT-Befehl wird immer eine Eingabe angefordert, die einer Variablen zugewiesen wird. Im Programm muß darum nach dem Befehlswort INPUT auch immer der Name der betreffenden Variablen stehen. Ist dies eine Zahlenvariable, so nimmt Ihr Genie auch nur Ziffern als Eingabe an und fordert Sie mit dem Wort REDO auf, die Eingabe zu wiederholen, wenn Sie versucht haben, einen Buchstaben einzutasten. Bei einer Zeichenkettenvariablen im INPUT-Befehl können sie alle Zeichen der Tastatur eingeben.

In einem INPUT-Befehl können auch mehrere Eingaben angefordert werden. Die Variablen, denen die Eingaben zugewiesen werden sollen, müssen dann durch Kommas getrennt sein. Aus diesem Grund darf das Komma und auch der Doppelpunkt innerhalb der Eingabe nicht verwendet werden, da sie vom Computer als Trennzeichen verwendet werden.

Wird z. B. in einer Zahlenfolge ein Komma oder Doppelpunkt statt des vorgeschriebenen Dezimalpunktes verwendet, übernimmt der Computer die Ziffern hinter dem Komma nicht und meldet EXTRA IGNORED. Folgt im INPUT-Befehl eine weitere Variable, wird der Wert nach dem Komma dieser zugewiesen.

Ein nach einer INPUT-Aufforderung eingegebener Doppelpunkt bewirkt in jedem Fall, daß alles, was danach kommt, nicht übernommen wird.

```
>10 INPUT A , B$
>20 PRINT A ; B$
>RUN
? 10.65
?? DM
  10.65 DM
READY
>■
```

Korrekte Eingabe

```
>RUN
? 10,65
  10 65
READY
>■
```

Zahlen nach dem Komma wurden der 2. Eingabe zugeordnet.

```
>RUN
? DM
REDO
? 123.50
?? DM
  123.5 DM
READY
>■
```

Fehler bei der Eingabe. (Es wurde versucht, als erstes einen String einzugeben).

```
>RUN
? 3 : 7
?? TORE
 3 TORE
READY
>■
```

Zahl nach dem Doppelpunkt wurde nicht übernommen.

Anführungsstriche können bei Zeichenketten vorn und hinten mit eingegeben werden. Sie werden dann aber nicht mit gespeichert. Stehen Anführungsstriche inmitten einer Kette, werden sie wie ein anderes Zeichen behandelt.

Wenn beim Lauf eines längeren Programms auf dem Bildschirm plötzlich ein Fragezeichen erscheint, weiß der Bediener möglicherweise gar nicht, was er eingeben soll, und wenn etwas Falsches eingegeben wird, kann das ganze Programm durcheinander geraten.

Bei älteren BASIC-Versionen mußte man darum vor jedem INPUT-Befehl noch einen PRINT-Befehl zur Erläuterung der angeforderten Eingabe schreiben. Ihr Genie erlaubt die Zusammenfassung einer Ausgabe auf dem Bildschirm mit einer Eingabe.

Hierzu ist es lediglich notwendig, die Worte, die zur Erläuterung der Eingabe auf dem Bildschirm geschrieben werden sollen, in Anführungsstrichen sofort hinter das Befehlswort INPUT zu setzen. Danach muß vor der ersten Variablen ein Semikolon stehen. Andere Trennzeichen sind nicht erlaubt.

Auf dem Bildschirm erscheint nach den erläuternden Worten zunächst ein Fragezeichen und dann die eingegebenen Zeichen.

Der Computer setzt in jedem Fall, auch wenn keine Eingabe erfolgt ist, den Programmlauf fort, wenn die RETURN-Taste gedrückt wird. Mit Hilfe einer „leeren Eingabe", bei der eine beliebige String-Variable, die sonst nicht gebraucht wird, verwendet werden kann, ist es möglich, das Programm in Wartestellung zu bringen.

```
>10 INPUT A , B$
>20 PRINT A ; B$
>RUN
? 333
?? "Bei Issus Keilerei"
 333 Bei Issus Keilerei
READY
>RUN
? 333
?? Das ist eine "Schnapszahl"
 333 Das ist eine"Schnapszahl"
READY
>■
```

Anführungsstriche werden nur in Zeichenketten übernommen, wenn sie im Innern stehen.

```
>10 INPUT "Betrag eingeben" ; A
>20 INPUT "2. Betrag eingeben" ; B
>30 S = A + B
>40 PRINT "Summe =" ; S
>RUN
Betrag eingeben? 342.67
2. Betrag eingeben? 186.45
Summe = 529.12
READY
>■
```

Zwischen „Erläuterung" und Variabler muß ein Semikolon stehen.

12. der GOTO-Befehl

Beim normalen Programmablauf wird ein Befehl nach dem anderen mit der niedrigsten Zeilennummer, die im Programm angesprochen wurde, beginnend, vom Computer abgearbeitet, bis die höchste im Programm stehende Zeilennummer erreicht ist. Von dieser rein sequentiellen Befehlsverarbeitung kann man durch den Sprung-Befehl mit dem Befehlswort GOTO abweichen. Dieser Befehl ist sehr einfach aufgebaut, denn hinter das Befehlswort ist nur die Zeilennummer zu setzen, in der das Programm fortgesetzt werden soll.

Mit Hilfe des GOTO-Befehls kann man so zum Beispiel dafür sorgen, daß ein Programmteil mehrfach durchlaufen wird. Das Programm läuft dann in einer Schleife solange, bis es durch Betätigung der BREAK-Taste abgebrochen wird.

Bei einer Schleife wird ein Rücksprung programmiert. Man kann mit GOTO aber natürlich auch vorwärts, also in Richtung höherer Zeilennummern springen. Voraussetzung für die Anwendung dieses Befehls ist aber, daß die nach GOTO folgende Zeilennummer auch mit einem BASIC Befehl belegt ist, sonst meldet das Genie UL ERROR oder UNDEFINED LINE und bricht dann den Programmlauf ab.

Es sei bereits hier darauf hingewiesen, daß man GOTO-Befehle, die einen unbedingten Sprung veranlassen, möglichst sparsam im Programm verwenden sollte, das heißt, sie sollen nur dort benutzt werden, wo dafür ein echtes Bedürfnis, zum Beispiel zum Aufbau einer Schleife besteht.

```
>10 INPUT A
>20 E = E + A
>30 PRINT , E
>40 GOTO 10
>RUN
? 43
                    43
? 84
                    127
? 57
                    184
?
                    241
? ■
```

Programm zur laufenden Addition eingegebener Zahlen, das in einer ewigen Schleife läuft und nur durch die BREAK-Taste abgebrochen werden kann.

Achtung! Betätigen Sie bei einem derartigen Programm niemals die <RETURN>-Taste ohne vorherige Eingabe einer Zahl, sonst wird wie beim Beispiel die zuletzt eingegebene Zahl nochmal addiert.

13. Der Entscheidungsbefehl

Die wohl wichtigsten Befehle in der gesamten Computertechnik, sind die Befehle, durch die bedingte Sprünge veranlaßt werden. Das bedeutet, daß die Sprünge im Programm nur ausgeführt werden, wenn eine vom Programmierer vorher definierte Bedingung vom Computer geprüft und als erfüllt anerkannt wird.

Für Entscheidungen kennt Ihr Genie den vielseitigen Befehl

IF . . . THEN . . . ELSE . . .

dessen englische Worte sich leicht ins Deutsche übersetzen lassen. Sie beschreiben gut die Wirkung dieses Befehls:

WENN	die nachfolgende Bedingung erfüllt ist
DANN	führe die nachfolgende Anweisung aus
SONST	arbeite mit der folgenden Anweisung weiter.

Hinter dem Befehlswort IF muß also immer etwas stehen, was der Computer prüfen kann. Es dürfen aber nur Bedingungen formuliert werden, auf die eindeutig nur mit JA oder NEIN geantwortet werden kann. Dies ist der Fall, wenn zwei Ausdrücke miteinander verglichen werden und hierzu dienen die Vergleichsoperatoren.

Grundsätzlich kann vor und hinter dem Vergleichsoperator entweder eine Variable, eine Konstante oder auch ein Rechenausdruck stehen. Es hat natürlich wenig Sinn, vor und hinter den Operator jeweils eine Konstante zu setzen, denn das Ergebnis eines derartigen Vergleichs wäre von vorneherein bekannt. An einer Seite sollte also immer eine Variable, deren Wert noch nicht bekannt ist, stehen, die dann vom Computer überprüft werden soll.

Nach dem Wort THEN können ein oder auch mehrere Befehle stehen, die der Computer ausführt, wenn der Vergleich als richtig oder, wie man auch sagt, „wahr" erkannt wurde. Sehr oft ist es üblich, an dieser Stelle einen unbedingten Sprungbefehl einzusetzen, weil man dann von der Zeilennummer an, in die gesprungen wurde, beliebig viele Befehle im Programm anordnen kann.

IF	wenn A
A = B	gleich B
A <> B	ungleich B
A > B	größer B
A < B	kleiner B
A >= B	größer oder gleich B
A <= B	kleiner oder gleich B
THEN	ist, dann

Die BASIC Vergleichsoperatoren

A = 5 , B = 6

A = B	unwahr
A <> B	wahr
A > B	unwahr
A < B	wahr
A >= B	unwahr
A <= B	wahr

Ergebnisse von Vergleichen

```
>10 INPUT "2 Zahlen" ; A , B
>20 IF A = B THEN PRINT "beide gleich" : END
>30 IF A < B THEN 50
>40 PRINT "1. Zahl größer" : GOTO 10
>50 PRINT "2. Zahl größer" : GOTO 10
>RUN
2 Zahlen? 6,2
1. Zahl größer
2 Zahlen? 3,8
2. Zahl größer
2 Zahlen? 0.001,—10
1. Zahl größer
2 Zahlen? 33,33
beide gleich
READY
>■
```

Programm zum Vergleich zweier Zahlen.

Die Verwendung des Befehlswortes ELSE ist nicht zwingend vorgeschrieben. Sie können Entscheidungsbefehle auf Ihrem Genie also auf zwei verschiedene Arten programmieren:

- Wenn das Wort ELSE im Befehl steht, wird bei Nichterfüllung der Vergleichsbedingungen das Programm mit dem oder den diesem Wort folgenden Befehlen fortgesetzt,
- wird das Wort ELSE nicht verwendet, führt der Computer bei als unwahr erkannter Bedingung die im Programm nächstfolgende Zeile aus.

Bei der Verwendung der IF ... THEN-Befehle müssen Sie immer daran denken, daß die beiden Zweige des Programms wieder zusammenlaufen müssen.

Die Beispiele zeigen die gleiche Aufgabe auf dreierlei Weise programmiert. Eine eingebene Zahl wird dahingehend überprüft, ob sie positiv oder negativ ist. Im letzten Fall wird sie mit —1 multipliziert, damit immer mit einem positiven Wert weitergearbeitet werden kann.

Die Erfüllung dieser Forderung ist einfach, wenn in den beiden Zweigen, die je nach der Entscheidung zu durchlaufen sind, nur wenige Befehle nötig sind. Sie können diese dann in den Entscheidungsbefehl mit einbauen und das Programm läuft auf jeden Fall mit der nachfolgenden Zeile weiter. (Beispiel 1)

Bei allen Entscheidungsbefehlen bietet Ihnen Ihr Genie noch die Erleichterung, daß das Wort GOTO unmittelbar nach THEN oder ELSE auch weggelassen werden kann. Es genügt die Einsetzung der Zeilennummer, zu der gesprungen werden soll.

Wenn der Entscheidungsbefehl ohne ELSE verwendet wird, muß als letzter Befehl hinter THEN ein Sprung-Befehl stehen, mit dem die Befehle der NEIN-Entscheidung übersprungen werden. (Beispiel 3)

Beispiel 1

```
>10 INPUT "Beliebige Zahl" ; A
>20 IF A > 0 THEN PRINT "+" A ELSE PRINT A : A = A ★ —1
>30 PRINT "Zahl ist jetzt immer positiv" ; A
>RUN
  Beliebige Zahl? 5
+ 5
Zahl ist jetzt immer positiv 5
READY
>RUN
  Beliebige Zahl? —7
—7
Zahl ist jetzt immer positiv 7
READY
>■
```

Beispiel 2

```
>10 INPUT "Beliebige Zahl" ; A
>20 IF A> 0 THEN PRINT "+" ; A : GOTO 40
>30 PRINT A : A = A ★ —1
>40 PRINT "Zahl ist jetzt immer positiv" ; A
>RUN
Beliebige Zahl? —7
—7
Zahl ist jetzt immer positiv 7
READY
>■
```

Können die nach THEN erforderlichen Befehle nicht im Entscheidungsbefehl untergebracht werden, verwendet man einen Sprung-Befehl zu einer höheren Zeilennummer und kann danach beliebig viele Befehle schreiben, die zur Wirkung kommen, wenn der Vergleich das Ergebnis JA hatte. Man muß dann allerdings die Befehle, die bei NEIN gültig sind, auch mit einem GOTO-Befehl abschließen , um die JA-Befehle zu überspringen. (Beispiel 3)

```
>10 INPUT "Beliebige Zahl" ; A
>20 IF A > 0 THEN 50
>30 PRINT A : A = A ★ —1
>40 GOTO 60
>50 PRINT "+" ; A
>60 PRINT "Zahl ist jetzt immer positiv" ; A
>RUN
Beliebige Zahl? 4
+ 4
Zahl ist jetzt immer positiv 4
READY
>■

>RUN
Beliebige Zahl —8
—8
Zahl ist jetzt immer positiv 8
READY
>■
```

Beispiel 3

Mit den Vergleichoperatoren können Sie in Genie-BASIC nicht nur Zahlen, sondern auch Zeichenketten miteinander vergleichen. Mit Hilfe des Gleichheitszeichens läßt sich also leicht feststellen, ob zwei Strings genau gleich sind.

Mit Hilfe der „größer" und „kleiner" Operatoren lassen sich Zeichenketten sogar nach dem Alphabet sortieren. Man benutzt hier die Tatsache, daß auch Buchstaben intern im Computer als Zahlen abgespeichert werden. Für die Speicherung wird der ASCII-Code verwendet, den Sie in der Genie-Befehlstabelle finden. Da in diesem Code dem Buchstaben A eine kleinere Zahl zugeordnet ist, als dem im Alphabet nachfolgenden Buchstaben, bedeutet der Operator „kleiner" in diesem Fall „im Alphabet vorher stehend".

```
10 X$="HAUS"
11 Print X$
20 Y$="HANS"
30 IF X$<Y$ THEN A$=X$ : B$=Y$ : GOTO 50
40 A$=Y$ : B$=X$
50 PRINT A$ ; "im Alphabet vor" ; B$
HANS im Alphabet vor HAUS
```

Im Beispiel werden die beiden String-Variablen X$ und Y$ verglichen. Die Variable mit dem kleineren Wert wird A$ und diejenige mit dem größeren Wert B$ zugewiesen. Beides wird dann mit Erläuterung gedruckt.

14. Aufbau von Schleifen

Schleifen, also gewollte Wiederholungen einzelner Programmteile, sind ein besonders wichtiges Hilfsmittel für eine effektive Programmierung. Wenn Sie BASIC lernen wollen, ist es darum sehr wichtig, daß Sie das Prinzip beim Aufbau von Schleifen richtig verstehen.

Es wurde schon von der ewigen Schleife, die nur durch Betätigung der BREAK-Taste abgebrochen werden kann, gesprochen, aber diese wird man im Programm wohl nur selten verwenden. Im allgemeinen muß ein gezielter Abbruch der Schleife mit programmiert werden.

Eine Schleife muß immer die folgenden Befehle enthalten:
- Einen Befehl zum Rücksprung an den Anfang der Schleife,
- einen Befehl, der entscheidet, ob die Bedingung zum Abbruch erfüllt ist,
- einen Befehl, der den Wert für die Abbruchbedingung bei jedem Schleifendurchgang ändert
- und natürlich die Befehle, die in der Schleife verarbeitet werden sollen.

Im Grundsatz kann man Schleifen auf zwei veschiedene Weisen aufbauen:
- Man kann den Veränderungsbefehl an den Anfang setzen, danach kommen die eigentlichen Schleifenbefehle und am Schluß wird geprüft, ob die Abbruchsbedingung erfüllt ist und an den Schleifenanfang zurückgesprungen, wenn dies nicht zutrifft. (Beispiel 1)
- Man kann aber auch die Schleifenbefehle an den Anfang schreiben, dann den Entscheidungsbefehl setzen und danach erst den Veränderungsbefehl und den Sprungbefehl setzen. Meist braucht man noch einen Befehl vor der Schleife zur Festlegung der Anfangsbedingungen. (Beispiel 2)

In den Beispielen sind Schleifen programmiert, in denen die Zahl 7 mit den Zahlen 1 bis 7 multipliziert wird.

```
>20 A = A + 1
>30 B = B ★ 7
>40 PRINT B ;
>50 IF A = 7 THEN END
>60 GOTO 20
>RUN
 7  14  21  28  35  42  49
READY
>■
```

Beispiel 1

```
>10 A = 1
>30 B = A ★ 7
>40 PRINT B ;
>50 IF A = 7 THEN END
>60 A = A + 1
>70 GOTO 30
>RUN
 7  14  21  28  35  42  49
READY
>■
```

Beispiel 2

Sie erkennen, daß in den Beispielen die Variable A bei jedem Schleifenlauf verändert wird und daß die Entscheidung, ob die Schleife abgebrochen wird, von dem Wert von A abhängt. Man bezeichnet A darum auch als Lauf- oder Zählvariable.

Beim Aufbau von Schleifen müssen Sie den Wert dieser Laufvariablen immer im Auge behalten. Wenn die Schleifen nicht genau wie in den Beispielen aufgebaut sind, können zu viel oder zu wenig Schleifendurchgänge stattfinden.

Es muß auch noch darauf hingewiesen werden, daß es manchmal zweckmäßig ist, im Entscheidungsbefehl eine Ungleich-Bedingung (<>) zu formulieren und dann solange zum Schleifenanfang zu springen, wie diese Bedingung noch erfüllt ist.

Bei Ihrem Genie gibt es aber noch eine besondere Kombination von Befehlsworten, mit denen Schleifen programmiert werden können. Es sind die FOR... NEXT-Anweisungen. Die vollständig wie folgt lauten:

Am Beginn der Schleife FOR V=X TO Y STEP Z
am Ende der Schleife NEXT V

Die Anwendung dieser Befehle ist ganz einfach. Zu Beginn der Schleife muß bestimmt werden, welche Variable als Laufvariable dienen soll. Diese wird bei jedem Schleifendurchgang verändert. Die Laufvariable steht hinter dem ersten Befehlswort FOR vor dem Gleichheitszeichen. Dann kommt der Anfangswert der Laufvariablen, das heißt also der Wert, mit dem die Schleife beginnen soll, und nach dem Befehlswort TO der Endwert, also der Wert nach dessen Erreichen die Schleife abzubrechen ist. Soll der Schritt, um den die Laufvariable bei jedem Schleifendurchgang zu verändern ist, nicht +1 betragen, so kann der gewünschte Schritt nach dem Befehlswort STEP angegeben werden.

```
>10 A = 1
>20 IF A = 7 THEN END
>30 B = A ★ 7
>40 PRINT B ;
>50 A = A + 1
>60 GOTO 20
>RUN
 7  14  21  28  35  42
READY
>■
```

Diese Schleife wird nur 6 mal durchlaufen, da der Abbruchbefehl vor den Schleifenbefehlen liegt.

```
>10 FOR A = 1 TO 10
>20 PRINT A ;
>30 NEXT A
>40 END
>RUN
 1  2  3  4  5  6  7  8  9  10
READY
>■
```

Einfache FOR...NEXT Schleife mit Ausdruck der Laufvariablen.

```
>20 A = A + 1
>30 B = A ★ 7
>40 PRINT B ;
>50 IF A <> 7 THEN 20
>60 END
>RUN
 7  14  21  28  35  42  49
READY
>■
```

Schleife wie Beispiel 1 aber mit Ungleich-Abfrage

```
>10 B = 6
>20 FOR A = B TO 2 ★ B + B / 2
>30 PRINT A ; : NEXT
>RUN
 6  7  8  9  10  11  12  13  14  15
READY
>■
```

FOR... NEXT Schleife mit Berechnung einer Grenze.

Als Anfangs- und Endwerte für die Laufvariable und ebenso auch für den Schritt können beliebige positive und negative Zahlen aber auch Rechenausdrücke eingesetzt werden. Das nebenstehende Beispiel zeigt einige hintereinanderstehende FOR... NEXT-Schleifen, bei denen immer nur die Laufvariable ausgedruckt wird. Beachten Sie das Semikolon nach dem Print-Befehl, das dafür sorgt, daß nach diesem Befehl kein Zeilenvorschub erfolgt. Damit aber vor dem Ausdruck der nächsten Schleife ein Zeilenvorschub stattfindet, muß jedem NEXT- ein leerer PRINT-Befehl folgen.

Beim Genie können Sie die Laufvariable nach NEXT weglassen, wenn nur eine FOR... NEXT-Schleife geöffnet ist. Dies bedeutet, daß Sie auch mit ineinander verschachtelten Schleifen arbeiten können, dann aber die gültige Laufvariable hinter NEXT einsetzen müssen. Dabei darf die Verschachtelung nur so aufgebaut werden, wie es das untenstehende Bild zeigt.

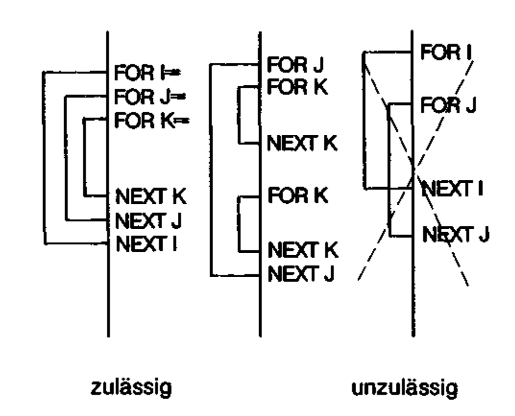
FOR... NEXT-Schleifen arbeiten bei Ihrem Genie so wie im Beispiel 1 auf Seite 35 dargestellt. Die Abfrage, ob die Schleifenbedingung erfüllt ist, erfolgt erst am Schluß. Jede Schleife wird also mindestens einmal durchlaufen.

Einsprünge in FOR... NEXT-Schleifen sind grundsätzlich nicht erlaubt. Wenn man dies versucht, meldet der Computer, wenn er auf ein NEXT ohne vorausgegangenes FOR trifft, NF ERROR oder NEXT WITHOUT FOR ERROR und bricht das Programm ab.

Bedingte Aussprünge aus Schleifen sind zulässig, können aber leicht zu Fehlern führen, wenn man nicht den Wert der Laufvariablen immer im Auge behält.

```
>10 B = 6
>20 FOR A = 20 TO 8 STEP —2
>30 PRINT A ; : NEXT : PRINT
>40 FOR A = .1 TO 1.4 STEP .2
>50 PRINT A ; : NEXT : PRINT
>60 FOR A = B TO 2★B + B / 2 — 1
>70 PRINT A ; : NEXT : PRINT
>RUN
 20  18  16  14  12  10  8
 .1  .3  .5  .7  .9  1.1  1.3
 6  7  8  9  10  11  12  13  14
READY
>■
```

Verschiedene FOR... NEXT Schleifen.



Verschachtelte Schleifen

```
>10 FOR A = 0 TO 0
>20 PRINT "Schleife durchlaufen"
>30 NEXT
>RUN
Schleife durchlaufen
READY
>■
```

Jede FOR... NEXT Schleife wird mindestens 1 mal durchlaufen.

15. Ein Programm entsteht

Nun haben Sie die fünf wichtigsten BASIC-Befehle kennengelernt und sich mit Hilfe der kleinen Übungsprogramme mit Ihrem Genie vertraut gemacht. Jetzt sollen Sie lernen, wie man bei der Entwicklung eines für die Praxis nützlichen Programms vorgehen kann, denn diese 5 Befehle genügen wirklich, um viele Aufgaben zu lösen.

Nehmen wir an, Sie müssen das Gewicht verschiedener Eisenrohre berechnen. Das geht zwar auch mit Hilfe eines Taschenrechners, aber Ihr Genie macht das viel komfortabler.

Wenn man etwas berechnen will, muß man die dafür anzuwendende Formel kennen. Das Gewicht eines Körpers ist immer
Gewicht = Grundfläche mal Länge mal spez. Gewicht
Beim Rohr entspricht die Grundfläche der Fläche eines Kreisringes, für die folgende Formel gilt:

$$\text{Kreisringfläche} = \frac{D^2 \cdot \pi}{4} - \frac{d^2 \cdot \pi}{4}$$

Nun können Sie sich direkt an Ihr Genie setzen und mit dem Programmieren beginnen, denn die Programme werden in BASIC im Dialogbetrieb mit dem Computer geschrieben. Trotzdem ist es aber zweckmäßig, wenn man sich vorher noch Gedanken über die zu benutzenden Variablennamen macht und diese getrennt aufschreibt.

Hier werden zunächst gebraucht:
GE für das auszurechnende Gewicht
FL für die Kreisringfläche
DA für den Außendurchmesser des Rohrs
DI für den Innendurchmesser des Rohrs
LG für die Rohrlänge

Nun fängt man zweckmäßigerweise bei den ersten Befehlen mit einer etwas größeren Zeilennummer an, damit später noch Befehle davor gesetzt werden können. Für diese Aufgabe werden 3 Eingaben gefordert, für die INPUT-Befehle in den Zeilen 100 und 110 geschrieben werden. (Beispiel 1)

```
>100 INPUT LG
>110 INPUT DA,DI
>120 FL = DA★DA★3.1415 / 4 — DI★DI★3.1415 / 4
>130 GE = FL ★ LG ★ 7.8
>140 PRINT GE
>RUN
? 100
? 2.54
?? 1.97
  1574.79
READY
>■
```

Beispiel 1

Die beiden Teile der Rechenformel könnten an sich in einer Programmzeile untergebracht werden, wenn man die Formeln zusammenfaßt. Einfacher und besser verständlich wird das Programm aber, wenn zuerst die Fläche und dann das Gewicht gerechnet wird.

In Zeile 120 wird der Variablen FL das Ergebnis des rechts neben dem Gleichheitszeichen stehenden Rechenausdrucks zugewiesen. Der Ausdruck entspricht der Formel, die nur nach den BASIC-Regeln geschrieben wurde. Die Zahl pi ist als Konstante eingesetzt. Das gleiche gilt für die Zeile 130, wo der Rechenausdruck für das Gewicht mit der Konstanten 7.8 für das spezifische Gewicht von Eisen steht.

Nun kommt noch der PRINT-Befehl für den Ausdruck des Gewichts hinzu und dann können Sie mit diesem Programm schon beliebige Rohrgewichte ausrechnen.

Damit ist das Programm aber noch nicht fertig. Wir wollen auch schon bei dieser kleinen Aufgabe alle Vorteile, die ein Computer gegenüber einem Taschenrechner bietet, ausnutzen.

In erster Linie muß sich ein Programm selbst erklären, denn sonst weiß man bald nicht mehr, was beim Erscheinen des Fragezeichens einzugeben ist. Die INPUT-Befehle erhalten darum eine Erläuterung. Dann muß klar gezeigt werden, was der Computer ausgerechnet hat. Hierfür wird der PRINT-Befehl entsprechend ergänzt. Da die Zahl pi mehrfach gebraucht wird, ist es ratsam, den Zahlenwert von pi einer Variablen genannt PI zuzuweisen und einen entsprechenden Befehl dem bisherigen Programm voranzustellen. Wenn dies geschehen ist, sieht das Programm wie im Beispiel 2 dargestellt aus.

```
>90 PI = 3.14159
>100 INPUT"Rohrlänge in cm" ; LG
>110 INPUT "Außen- und Innendurchmesser" ; DA,DI
>120 FL = DA ★ DA ★ PI / 4 — DI ★ DI ★ PI / 4
>130 GE = FL ★ LG ★ 7.8
>140 PRINT " Das Rohr wiegt" ; GE ; "g"
>run
Rohrlänge in cm? 100
Außen- und Innendurchmesser? 2.54,1.97
Das Rohr wiegt 1574.84 g
READY
>■
```

Beispiel 2

Einer der Vorteile von BASIC liegt darin, daß es verhältnismäßig einfach ist, ein Programm zu ändern und zu ergänzen. So soll auch hier gezeigt werden, wie dieses Programm noch weiter ausgebaut werden kann. Es soll so erweitert werden, daß auf Wunsch auch eine ganze Tabelle von Rohrgewichten erstellt werden kann.

Hierfür ist es zunächst notwendig, dem Benutzer eine Auswahlmöglichkeit für die Einzelberechnung oder die Tabelle zu geben. Zu diesem Zweck werden die Befehle in den Zeilen 20—50 hinzugefügt. Dabei sorgt der Entscheidungsbefehl in Zeile 50 für eine erneute Abfrage, wenn weder E noch T eingegeben wurden.

Entsprechend dem Befehl in Zeile 40 beginnt das Tabellenprogramm in Zeile 200 mit INPUT-Befehlen. Für die Berechnung und die Ausgabe der Tabellenwerte wird eine FOR... NEXT-Schleife verwendet, die die eingegebenen Variablen verarbeitet. Hierbei ist darauf zu achten, daß der kleinste Durchmesser vor dem TO steht, da mit positivem Schritt gearbeitet wird. Zeile 260 muß etwas geändert werden, da als Länge konstant 100 cm angenommen werden und das Gewicht in kg ausgegeben werden soll. Durch den PRINT-Befehl werden die Durchmesser und das Gewicht in zwei Spalten gedruckt.

Das Programm erhält nun noch eine Überschrift. Die Wertzuweisung an PI, die für beide Programmteile gilt, muß an den Anfang gesetzt werden, und in Zeile 150 muß ein END-Befehl kommen.

Jetzt kann das Programm ausprobiert werden, und dabei entdeckt man noch zwei kleine Fehler. Es fehlt eine Tabellenüberschrift, die vor die Schleife in Zeile 225 programmiert wird. Außerdem kann es vorkommen, daß der letzte Schleifendurchgang wegfällt, weil der Computer beim Rechnen GD abgerundet hat. Dies wird durch Hinzufügen einer Addition von 0.0001 zu GD in Zeile 230 ausgeglichen.* Das fertige Programm ist als Beispiel 3 auf Seite 41 gezeigt.

*Dies hängt damit zusammen, daß der Computer intern binär rechnet. Nicht jeder Dezimalbruch hat ein genaues binäres Äquivalent, ebensowenig wie z. B. 0.3333333 genau 1/3 ist.

```
20 PRINT"Einzelrohr ( E ) oder Tabelle ( T )
30 INPUT"Kennbuchstabe" ; K$
40 IF K$ = "T" THEN 200
50 IF K$ <> "E" THEN 20

200 INPUT"Wandstärke" ; WA
210 INPUT"größter und kleinster Außendurchm." ; GD,KD
220 INPUT"Stufung" ; ST
230 FOR DA = KD TO GD STEP ST
240 FI = DA — WA ★ 2
250 FL = ( DA★DA — DI★DI ) ★ PI / 4
260 GE = FL ★ 100 ★ 7.8 ★ .001
270 PRINT DA ; " / " ; DI, GE
280 NEXT

5 CLS
10 PRINT"Rohrgewichte berechnen"
15 PI=3.14159

150 END

'Änderung des Programms Beispiel 2
```

```
Rohrgewichte berechnen
Einzelrohr ( E ) oder Tabelle ( T )
Kennbuchstabe? T
Wandstärke? .25
größter und kleinster Außendurchm.?. 2,1
Stufung? .2
  1 / .5          .459458
  1.2 / .7        .58198
  1.4 / .9        .704502
  1.6 / 1.1       .827024
  1.8 / 1.3       .949546
READY
>■
```

Lauf des Programms nach Änderungen.

```
5 CLS
10 PRINT"Rohrgewichte berechnen"
15 PI=3.14159
20 PRINT"Einzelrohr ( E ) oder Tabelle ( T )
30 INPUT"Kennbuchstabe" ; K$
40 IF K$ = "T" THEN 200
50 IF K$ <> "E" THEN 20
100 INPUT"Rohrlänge in cm" ; LG
110 INPUT"Außen- und Innendurchmesser" ; DA,DI
120 FL = DA ★ DA ★ PI / 4 — DI ★ DI ★ PI / 4
130 GE = FL ★ LG ★ 7.8
140 PRINT "Das Rohr wiegt" ; GE ; "g"
150 END
200 INPUT"Wandstärke" ; WA
210 INPUT"größter und kleinster Außendurchm." ; GD,KD
220 INPUT"Stufung" ; ST
225 PRINT"DA / DA ( cm )","Gewicht / M ( KG )"
230 FOR DA = KD TO GD +.0001 STEP ST
240 DI = DA — WA ★ 2
250 FL = ( DA★DA — DI★DI ) ★ PI / 4
260 GE = FL ★ 100 ★ 7.8 ★ .001
270 PRINT DA ; " / " ; DI, GE
280 NEXT
```

```
Rohrgewicht berechnen
Einzelrohr ( E ) oder Tabelle ( T )
Kennbuchstabe? T
Wandstärke? .4
größter und kleinster Außendurchm.? 4.2,3
Stufung? .2
1DA / DI ( cm )   Gewicht / m ( kg )
  3 / 2.2         2.54846
  3.2 / 2.4       2.74449
  3.4 / 2.6       2.94053
  3.6 / 2.8       3.13656
  3.8 / 3         3.3326
  4 / 3.2         3.52863
  4.2 / 3.4       3.72467
READY
>■
```

Beispiel 3

16. Unterprogramme

Auch in BASIC kann man — wie in allen anderen Programmiersprachen — die Unterprogrammtechnik anwenden. Man versteht hierunter eine Programmierung, bei der gewisse, öfter gebrauchte Teile eines Programmes nur einmal als sogenanntes Unterprogramm geschrieben werden, das dann beliebig oft innerhalb des Gesamtprogramms aufgerufen werden kann.

Unterprogramme werden mit den Befehlsworten GOSUB mit nachfolgender Zeilennummer aufgerufen. Die Zeilennummer muß den ersten Befehl des betreffenden Unterprogramms enthalten. Der Computer merkt sich die nach dem GOSUB-Befehl folgende Zeilennummer und setzt dort das Programm fort, wenn er im Unterprogramm den Beendigungsbefehl RETURN findet.

Meist ist es zweckmäßig, die Unterprogramme an den Schluß des Programms zu legen. Man muß dann allerdings dafür sorgen, daß das Hauptprogramm durch einen END-Befehl abgeschlossen wird, weil sonst das erste Unterprogramm nach dem Hauptprogramm anläuft und mit der Fehlermeldung RG oder RETURN WITHOUT GOSUB beendet wird, wenn der Befehl RETURN gefunden wird.

Unterprogramme können viel dazu beitragen, daß das Programm übersichtlich bleibt. Wenn mehrere Blöcke in einem übergeordneten Hauptprogramm zusammengefaßt werden sollen, so ist hierfür der GOSUB-Befehl besser als GOTO-Befehle.

In BASIC werden in den Unterprogrammen die gleichen Variablen verwendet wie im Hauptprogramm, so daß die Übernahme von Daten von Unterprogrammen in das Hauptprogramm problemlos ist.

```
>10 A = 3 : B = 7
>20 E1 = 100 / A
>30 GOSUB 300
>40 PRINT E1, E
>50 E1 = A / B
>60 GOSUB 300
>70 PRINT E1, E
>100 END
>300 E = INT ( E1 * 100 + .5 ) / 100
>310 RETURN
>RUN
 33.3333        33.33
 .428571        .43
READY
>■
```

Um unnötige Nachkommastellen abzuschneiden wird nach jedem Divisionsbefehl das Unterprogramm in 300 aufgerufen, in dem eine Rundung auf 2 Nachkommastellen programmiert ist.

17. Logische Verknüpfungen

In der Schaltungstechnik spielen logische Verknüpfungen eine große Rolle, und auch in BASIC gibt es zwei Befehle durch die jeweils zwei oder auch mehrere Bedingungen miteinander verknüpft werden können. Es gelten hierfür die Befehlsworte

AND   Bedingung erfüllt, wenn alle Teile „wahr“ sind
OR   Bedingung erfüllt, wenn eins der Teile „wahr“ ist

Mit Hilfe dieser beiden Befehlsworte können in einem IF . . . THEN-Befehl mehrere Bedingungen formuliert werden, zwischen denen dann eins der Befehlsworte stehen muß. Im Programm werden alle Bedingungen geprüft. Ist die Verknüpfung mit AND erfolgt, wird der Befehl nach THEN nur ausgeführt, wenn die vor und hinter AND stehende Bedingung erfüllt ist. Bei OR genügt es, wenn eine Bedingung von beiden erfüllt ist.

Bei der Anwendung dieser Befehlsworte muß man beachten, daß vor und hinter ihnen jeweils ein vollständiger Vergleich stehen muß. Das folgende Statement ist nicht zulässig

```
IF A > 10 AND < 100 THEN 250
```

Dafür muß geschrieben werden:

```
IF A > 10 AND A< 100 THEN 250
```

```
>10 INPUT A,B
>15 PRINT"Mit" ; A ; "und" ; B
>20 IF A > B AND 2★A > B THEN 80
>30 IF A >B OR 2★A > THEN 90
>40 PRINT "nicht erfüllt"
>50 END
>80 PRINT"beide Bedingungen erfüllt" : END
>90 PRINT"eine Bedingung erfüllt" : END
>RUN
? 10,12
Mit 10 und 12
eine Bedingung erfüllt
READY
>RUN
? 10,4
Mit 10 und 4
beide Bedingungen erfüllt
READY
>RUN
? 10,30
Mit 10 und 30
nicht erfüllt.
READY
>■
```

Für gewisse Programmieraufgaben ist es noch wichtig, zu wissen, daß Ihr Genie intern eine erfüllte Bedingung mit —1 und eine nicht erfüllte Bedingung mit 0 registriert. Man kann also statt

IF A = B AND C = D AND E = F THEN... auch schreiben
IF (A=B) + (B=D) + (E=F) = —3 THEN...

Wenn nur zwei Bedingungen erfüllt zu sein brauchen, wird statt —3 einfach —2 eingesetzt.

```
>10 INPUT A,B,C
>20 IF ( A=3 )+( B=4 )+( C=5 ) = —3 THEN 50
>30 IF ( A=3 )+( B=4 )+( C=5 ) = —2 THEN 60
>40 IF ( A=3 )+( B=4 )+( C=5 ) = —1 THEN 70
>45 PRINT"keine Bedingung erfüllt" : END
>50 PRINT"alle Bedingungen erfüllt" : END
>60 PRINT"2 Bedingungen erfüllt" : END
>70 PRINT"1 Bedingung erfüllt" : END
>RUN
? 3,4,5
alle Bedingungen erfüllt
READY
>RUN
? 3,4,6
2 Bedingungen erfüllt
READY
>RUN
? 3,5,6
1 Bedingung erfüllt
READY
>■
```

18. ON...GOTO oder GOSUB Befehle

Der IF ... THEN-Befehl läßt, wie erklärt wurde, immer nur zwei Möglichkeiten für die Fortführung des Programmes zu. Wenn es gewünscht ist, eine ganze Reihe von Möglichkeiten für die Fortsetzung eines Programms vorzusehen, können Sie hierfür den ON ... GOTO-Befehl verwenden. Die vollständige Form dieses Befehls lautet

ON Variable GOTO Zeilen Nr., Zeilen Nr., Zeilen Nr. ...

Findet der Computer diesen Befehl, so wandelt er zunächst die Varaible nach dem Wort ON in einen Ganzzahlwert um. Bei Ihrem Genie kann darum auch hier ein Rechenausdruck stehen. Ist das Ergebnis eine 1, so wird die erste nach dem Wort GOTO stehende Zeilennummer angesprungen. Bei 2 erfolgt der Sprung zur 2. Zeilennummer u.s.w. u.s.w. ... Man kann also mit einem ON ... GOTO-Befehl mehrfache Verzweigungen programmieren.

Statt GOTO kann in den Befehlen auch das Wort GOSUB eingesetzt werden. Es erfolgt dann der Einsprung in ein Unterprogramm und nach dessen Ende der Rücksprung zu der Zeile, die dem ON ... GOSUB-Befehl folgt.

Mit diesem Befehl können Sie leicht ein Menü, wie man ein Verzeichnis für mehrere Programmteile zur Auswahl nennt, abfragen. Nach der Anzeige des Menüs folgt ein INPUT-Befehl, der vom Benutzer die Eingabe einer Zahl fordert. Diese Zahl wird der Variablen zugewiesen, die nach ON im Verzweigungs-Befehl steht. Nach GOTO bzw. GOSUB folgen dann die Zeilennummern, in denen die einzelnen Programmteile beginnen.

Wenn Sie diesen Befehl anwenden, müssen Sie aber immer dafür sorgen, daß das Hauptprogramm nach Beendigung der Unterprogramme richtig weiterläuft. Im Beispiel sorgt der Sprung-Befehl in Zeile 140 dafür, daß das Menü neu angezeigt wird.

```
>90 PRINT"            SPIELAUSWAHL"
>100 PRINT "1 = Mühle","2 = Dame"
>110 PRINT"3 = Autorennen","4 = Sternkrieg"
>120 INPUT A
>130 ON A GOSUB 200,300,500,900
>140 PRINT : PRINT"ein anderes Spiel?" : GOTO 100
>200 PRINT"Programm für Mühle"
>290 RETURN
>300 PRINT"Programm für Dame"
>390 RETURN
>500 PRINT"Programm für Autorennen"
>800 RETURN
>900 PRINT"Programm für Sternkrieg"
>2000 RETURN
>RUN
            SPIELAUSWAHL
1 = Mühle         2 = Dame
3 = Autorennen    4 = Sternkrieg
? 2
Programm für Dame

ein anderes Spiel?
1 = Mühle         2 = Dame
3 = Autorennen    4 = Sternkrieg
? ■
```

19. Fehler und Fehlersuche

Einen nicht unerheblichen Teil der Arbeitszeit beim Programmieren muß man für die Fehlerbeseitigung einkalkulieren. Vorher kommt aber die Fehlersuche, und hierfür bietet Ihr Genie eine Reihe von Hilfen an.

Eine Fehlermeldung wird entweder als Fehlercode oder als englischer Text bei Disk-Betrieb auf dem Bildschirm ausgegeben, wenn der Computer bei der Abarbeitung eines Programms nicht weiterkommt, weil er einen Fehler entdeckt hat. Im wesentlichen geht es hier um zwei Fehlerarten.

Als *Syntaxfehler* werden diejenigen Fehler bezeichnet, bei denen der Computer ein in BASIC nicht zulässiges Sprachelement feststellt oder bei denen eine Regel dieser Sprache nicht eingehalten wurde.

Findet Ihr Genie einen Syntaxfehler, bricht es den Programmlauf ab und meldet, in welcher Zeile der Fehler gefunden wurde. Zur Erleichterung der Korrektur wird sofort das EDITOR-Programm aufgerufen, und es erscheint nach der Fehlermeldung die Nummer der fehlerhaften Zeile. Sie können diese nun sofort mit Hilfe der EDITOR-Befehle berichtigen.

Es gibt dann noch eine ganze Reihe *anderer Fehler,* die der Computer feststellen und melden kann. Diese Fehler können aber nicht durch Korrektur in einer Zeile behoben werden. Zum Teil müssen neue Befehle eingefügt werden oder es ist in anderer Weise in das Programm einzugreifen. Der Computer beendet darum bei diesen Fehlern nur den Programmlauf und meldet mit READY, daß er auf direkte Eingaben wartet.

20. Beschreibung der Fehlermeldungen

NF NEXT ohne FOR: Ein NEXT ohne entsprechendes FOR wurde benutzt. Dieser Fehler kann auch auftreten, wenn die NEXT-Variablen in verschachtelten Schleifen vertauscht wurden.

SN Syntax-Fehler: Ist normalerweise das Ergebnis von falscher Interpunktion, offenen Klammern, ungültigen Zeichen oder falsch geschriebenen Statements.

RG RETURN ohne GOSUB: Ein RETURN-Statement wurde gefunden, ohne daß ein entsprechendes GOSUB existiert.

OD Keine Daten mehr: Einem READ oder INPUT#-Statement stehen nicht mehr genug Daten zur Verfügung. Ein DATA-Statement kann vergessen worden sein oder auf dem Band sind schon alle Daten gelesen.

FC Illegaler Funktionsaufruf: Es wurde der Versuch gemacht, eine Operation mit ungültigen Parametern auszuführen. Beispiel: Quadratwurzel mit negativem Argument, negative Matrixdimension, Negativ- oder Null-Argumente für LOG, usw.... USR-Aufruf, ohne die Startadresse zu POKEN.

OV Überlauf: Ein berechneter Wert oder eine Eingabe ist zu groß oder zu klein. Der Computer kann ihn nicht mehr handhaben.

OM Kein Speicherplatz mehr: Der gesamte Speicherplatz ist entweder benutzt oder reserviert worden. Die Ursache können zu große Felddimensionen oder verschachtelte Sprungbefehle wie GOTO, GOSUB und FOR-NEXT sein. Es kann aber auch schlicht das Programm zu lang sein.

UL Undefinierte Zeile: Es wurde versucht, zu einer nicht existierenden Zeile zu springen.

BS Dimensionen aus ihrem Bereich: Es wurde versucht, ein Feldelement anzusprechen, dessen Dimensionen über der im DIM-Statement angegebenen liegt.

DD Redimensionierung: Es wurde versucht, ein Feld zu dimensionieren, das schon dimensioniert ist (mit DIM oder implizit). Guter Programmierstil ist es, DIM-Statements an den Programmanfang zu stellen.

/0 Division durch Null: Es wurde versucht, Null als Divisor zu benutzen.

ID Illegaler, direkter Einsatz: Es wurde versucht, das INPUT-Statement auf der Kommandoebene auszuführen.

TM Typ stimmt nicht überein: Es wurde versucht, in eine Nicht-Zeichenkettenvariable eine Zeichenkette zu schreiben oder umgekehrt.

OS Kein Platz mehr für Zeichenkette: Siehe CLEAR.

LS Zeichenkette zu lang: Eine Zeichenkette darf nur 255 Zeichen lang sein.

ST Stringformel zu komplex: Ein Zeichenkettenformel ist zu komplex, um sie zu handhaben.

CN Kann CONT nicht ausführen:. Ein CONT wurde eingegeben, obwohl kein Programm zum Weitermachen mehr vorhanden ist. Z. B. wenn das Programm mit END abgeschlossen wurde oder nach EDIT.

NR NO RESUME: Das Programmende wurde im Fehlererkennungsmodus erreicht.

RW RESUME ohne ERROR: Ein RESUME wurde gefunden, bevor ein ON ERROR GOTO ausgeführt wurde.

UE Nicht ausgebbarer Fehler: Es wurde versucht, einen Fehler mit nicht existierendem ERROR-Code mit ERROR zu simulieren.

MO Fehlender Operand: Es wurde versucht, eine Operation auszuführen, bei der einer der Operanden fehlte.

FD Defekte Datei: Die Dateneingabe von einer externen Quelle (Kassettenrekorder usw.) war falsch, in der falschen Reihenfolge usw.

21. Der EDITOR

Der Editor des Colour-Genie-Systems erlaubt seinem Benutzer, die Programmzeilen, die er geschrieben hat, zu korrigieren, ohne sie neu eintippen zu müssen. Die Notwendigkeit eines Editors wird besonders bei langen, komplexen Programmzeilen deutlich. In diesem Kapitel werden wir alle Editierungsfunktionen des Colour-Genie-Systems mit ihren Subkommandos besprechen. Wir werden unsere Erklärungen mit vielen Beispielen verständlicher machen. Dem Benutzer wird empfohlen, jedes Editierungskommmando zu probieren, bevor er sich an das Schreiben von Programmen begibt.

EDIT-Zeilennummer

Dieses Kommando bringt den Computer von der aktiven Kommandoebene in die Editierungsebene. Der Benutzer muß abgeben, welche Zeile er editieren will. Wird keine Zeilennummer angegeben, reagiert der Rechner mit einer UL Fehlermeldung.

Beispiel:

EDIT 100	eröffnet Editierung von Zeile 100.
EDIT .	eröffnet Editierung der gerade eingetippten Zeile.

Auf der Editierungsebene kann der Computer folgende Subkommandos ausführen:

<RETURN>-Taste

Wird die <RETURN>-Taste betätigt, wenn sich der Rechner im Editierungsmodus befindet, so speichert er alle vorher gemachten Änderungen ab und kehrt auf die aktive Kommandoebene zurück.

Leertaste

Im Editierungsmodus wird bei jeder Betätigung der Leertaste ein weiteres Zeichen in der editierten Zeile angezeigt. Wird eine Zahl n vor der Betätigung der Leertaste eingegeben, so werden n Zeichen der gerade editierten Zeile ausgegeben.

Nehmen wir an, wir haben folgende Zeile eingegeben:

```
AUTO 100
100 IF A = B THEN 150 : A = A + 1 : GOTO 100
```

Soll nun diese Zeile 100 editiert werden, so muß der Benutzer EDIT 100 eingeben.

>EDIT 100

Der Computer antwortet:

100 ■

Wird die Leertaste 12mal betätigt, so bewegt sich der Cursor 12 Zeichen nach rechts (Cursor = der Unterstrich, den der Computer hinter jedes eingegebene Zeichen setzt). Auf dem Schirm steht:

100 IF A = B THE ■

Man muß nicht für jedes Zeichen einzeln die Leertaste betätigen. Geben Sie 8, gefolgt von der Leertaste ein, erscheint folgende Zeile:

100 IF A = B THEN 150 : ■

Geben Sie die 20, gefolgt von der Leertaste ein, so erscheint schließlich die gesamte Zeile:

100 IF A = B THEN 150 : A = A + 1 : GOTO 100 ■

<←>-Taste

Bewegt den Cursor ein Zeichen zurück (entgegengesetzt zur Leertaste). Wird eine Zahl vor dem Betätigen der <←>-Taste eingegeben, so wird der Cursor um den Betrag dieser Zahl nach links bewegt. Entgegen der normalen Backspace-Funktion bleiben die so auf dem Schirm gelöschten Zeichen im Speicher erhalten.

Beispiel:

100 IF A = B THEN 150 A = A + 1 : GOTO 100 ■

Nach fünfmaligem Betätigen der BACKSPACE-Taste:

100 IF A = B THEN 150 : A = A + 1 : GOT ■

Geben Sie nun eine 10, gefolgt von der <←>-Taste ein, erscheint folgende Zeile:

100 IF A = B THEN 150 : A = A ■

Nochmals betont, um den Editierungsmodus zu verlassen, betätigen Sie die NEW LINE-Taste. Der Computer geht zurück auf die aktive Kommandoebene und auf dem Schirm steht:

>■

Wenn nun noch weitere Änderungen an Zeile 100 vorgenommen werden sollen, müssen Sie mit EDIT 100 in den Editierungsmodus zurück.

SHIFT <↑> Taste

Wird die SHIFT und die <↑>-Taste gleichzeitig betätigt, so stellt der Computer die Ausführung eines vorher gestarteten Subkommandos (zum Einsetzen von Zeichen H, I, X) ein. Der Computer befindet sich danach weiter auf der Editierungsebene und die Position des Cursors bleibt unverändert. Ein anderer Weg, um die Ausführung eines der Subkommandos zu beenden, ist die <RETURN>-Taste. Wird Sie betätigt, so kehrt der Computer auf die aktive Kommandoebene zurück.

H-Taste

„H" meint trenne ab (engl. hack) und setze ein. D. h. der Rest der Zeile wird gelöscht und an der momentanen Position des Cursors können Zeichen eingesetzt werden.

Beispiel:

100 IF A = B THEN 150 : A = A + 1 : GOTO 100 ■

Nehmen Sie an, Sie wollten A = A + 1 durch A = A + B ersetzen und das GOTO löschen.
Zunächst gehen Sie in den Editierungsmodus mit

EDIT 100

Die Zahl 25, gefolgt von der Leertaste, bringt den Cursor auf die 25. Position der Zeile:

100 IF A = B THEN 150 : A = A ■

Nun betätigen Sie die H-Taste und tippen + B ein. Dann betätigen Sie die <RETURN>-Taste und sind auf der aktiven Kommandoebene. Eine andere Möglichkeit, die Sie im Editierungsbereich bleiben läßt, ist das gleichzeitige Betätigen der SHIFT- und der <↑>-Taste. Geben Sie nun L für List ein. der Computer gibt einmal die gesamte Zeile aus und der Cursor geht an den Anfang der Zeile:

100 IF A = B THEN 150 : A = A + B
100 ■

Alles, was nicht ausgegeben wurde, ist effektiv gelöscht.

I-Taste

„I" meint Einsetzen (insert). Mit diesem Subkommando können Sie Zeichen an einer beliebigen Position einsetzen, ohne andere Teile ändern zu müssen.

Beispiel:
Nehmen Sie an, Sie wollen den Befehl „PRINT A" zwischen „A = A + 1" und „GOTO 100" in der Zeile 100 einsetzen. Die Zeile 100 soll vorher so aussehen:

100 IF A = B THEN 150 : A = A + 1 : GOTO 100

Mit der Leertaste bringen Sie den Cursor zur Position:

100 IF A = B THEN 150 : A = A + 1 : ■

Nun betätigen Sie die „I“-Taste und geben „PRINT A :“ ein. Mit der SHIFT- und <↑>-Taste verlassen Sie den Einsetzmodus (insert mode). Nun können Sie mit L die gesamte Zeile listen:

100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

100■

Mit NEW LINE können Sie dann auf die aktive Kommandoebene zurück.

X-Taste

„X“ meint setze am Ende der Zeile ein. Der Cursor wird an das Ende der Zeile bewegt und der Computer geht in den Einsetzmodus. Man kann nun am Ende der Zeile hinzufügen oder mit der <←>-Taste am Ende der Zeile Zeichen löschen.

Beispiel:

In den Editierungsmodus:

> EDIT 100
100 ■

Geben Sie nun „X“ (ohne <RETURN>-Taste) ein:

100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100 ■

An dieser Stelle können Sie neue Zeichen eingeben, schon existierende löschen und dieses Subkommando mit SHIFT und <↑> verlassen.

L-Taste

„L“ meint liste Zeile. Ist der Computer im Editierungsmodus, führt aber im Moment keines der Subkommandos H, I und X aus, so kann man mit dem L Subkommando den verbliebenen Teil der Zeile auf dem Schirm ausgeben.

Beispiel:
>EDIT 100
100 ■

Betätigen Sie die L-Taste:

100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

100 ■

In der zweiten Zeile können Sie editieren und dabei den Ursprungstext im Auge behalten.

A-Taste

„A" meint beginne neu (cancel and restart). Der Cursor wird wieder zum Zeilenanfang gebracht und alle vorher gemachten Änderungen werden gelöscht. Die Zeile bleibt in ihrem ursprünglichen Zustand.

E-Taste

Dieses Kommando bringt den Computer zurück auf die aktive Kommandoebene und trägt alle vorher gemachten Änderungen in der editierten Zeile ein. Der Computer darf kein Subkommando ausführen (H, I und X), wenn E eingegeben wird.

Q-Taste

Dieses Kommando bringt den Computer von der Editierungsebene zurück auf die aktive Kommandoebene, aber löscht alle vorher gemachten Änderungen. Die Zeile bleibt in ihrem ursprünglichen Zustand.

D-Taste

„D" meint löschen (delete). Wird vor D eine Zahl n eingegeben, so werden n Zeichen rechts von der momentanen Cursorposition gelöscht. Die gelöschten Zeichen werden in Ausrufezeichen eingeschlossen, um zu zeigen, daß sie von der Operation betroffen sind.

Beispiel:
100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

Schalten Sie auf die Editierungsebene und bewegen Sie den Cursor mit der Leertaste in folgende Position:

100 IF A = B THEN 150 : A = A + 1 ■

Nun geben Sie 15 D ein (zerstöre 15 Zeichen—:

100 IF A = B THEN 150 : A = A + 1 ! : PRINT A : GO !

Benutzen Sie L, um die gesamte Zeile zu listen:

100 IF A = B THEN 150 : A = A + 1 ! : PRINT A : GO ! TO 100

100 ■

Ein weiteres List bringt folgendes auf den Schirm:

100 IF A = B THEN 150 : A = A + 1 TO 100

100 ■

Um das „TO 100" zu löschen, benutzen Sie die X-Taste und die <←>-Taste. Das Endergebnis sollte sein:

100 IF A = B THEN 150 : A = A + 1

C-Taste

„C“ meint verändern (change). Wird eine Zahl n vor dem C eingegeben, so werden n Zeichen rechts von der momentanen Cursorposition verändert. Wird die Zahl nicht angegeben, so wird nur ein Zeichen verändert.

Beispiel:
100 IF A = B THEN 150 : A = A + 1

Wollen Sie die 150 in eine 230 ändern, so schalten Sie den Editierungsmodus (EDIT 100) ein und bewegen den Cursor mit der Leertaste in folgende Position:

100 IF A = B THEN ■

Geben Sie nun ein 2C (ändere 2 Zeichen), gefolgt von 23 (neue Daten). Dann verlassen sie das C-Subkommando mit SHIFT und <↑>. Listen Sie die Zeile mit L:

100 IF = B THEN 230 : A = A + 1

100 ■

nSc

Dieses Kommando sucht (search) nach dem n'ten Vorkommen des Zeichens c in der gerade editierten Zeile und setzt den Cursor an dessen Position. Wird n nicht angegeben, so wird nach dem ersten Auftreten des Zeichens c gesucht. Wird das Zeichen c nicht gefunden, so steht der Cursor am Ende der Zeile. Wie bei den anderen Subkommandos, beginnt der Rechner an der momentanen Cursorposition zu suchen.

Beispiel:
100 IF A = B THEN 230 : A = A + 1

Schalten Sie auf den Editierungsmodus (> EDIT 100):

100 ■

Nun geben Sie 2S = ein, um dem Computer mitzuteilen, er soll nach dem zweiten Auftreten des Zeichens „=“ suchen:

100 IF A = B THEN 230 : A ■

Nun kann ein anderes Subkommando aktiviert werden, um die so aufgefundenen Daten zu verändern. Etwa: Geben Sie H und danach = A + 2 (neue Daten) ein:

100 IF A = B THEN 230 : A = A + 2 ■

nKc

Dieses Kommando löscht alle Zeichen bis zum n-ten Vorkommen des Zeichens c und setzt den Cursor an diese Stelle.

Folgendes Beispiel:

100 IF A = B THEN 230 : A = A + 2

Schalten Sie auf den Editierungsmodus (> EDIT 100):

100 ■

Nun geben Sie ein 1K:, um dem Computer mitzuteilen, er solle nach dem ersten Auftreten des Zeichens Doppelpunkt „:“ suchen und solle alles löschen, was vor ihm in der Zeile steht:

100 ! IF A = B THEN 230 !

Soll der Doppelpunkt noch dazu gelöscht werden, geben Sie D ein:

100 ! IF A = B THEN 230 ! ! : !

Um das Ergebnis zu betrachten, geben Sie L ein: (2 mal):

100 A = A + 2

100 ■

22. Aktive Befehle

Wenn das System zusammengestellt und der Computer eingeschaltet wird, so müßte sich der Benutzer auf der aktiven Befehlsebene befinden. Normalerweise erkennt man das daran, daß das Wort „READY“, gefolgt von einem „>“-Zeichen in der nächsten Zeile in der oberen, linken Ecke des Schirms (Monitor oder Fernsehgerät) erscheint. Diese Anzeige nennen wir von nun an Fertigmeldung.

Nun können Sie eines der folgenden Kommandos eingeben:

AUTO	EDIT	SYSTEM
CLEAR	LIST	TROFF
CONT	NEW	TRON
DELETE	RUN	LLIST

Wir werden diese Kommandos im einzelnen diskutieren. Bitte beachten Sie, daß alle Angaben in Klammern optional sind; angegeben werden können, aber nicht müssen. Beispiel AUTO (Zeilennummer, Inkrement) Das konkrete Kommando sieht so aus:

AUTO 10,5   oder AUTO oder AUTO 10

Wollen Sie z. B. Ihr gesamtes Programm auflisten, so geben Sie LIST ein.

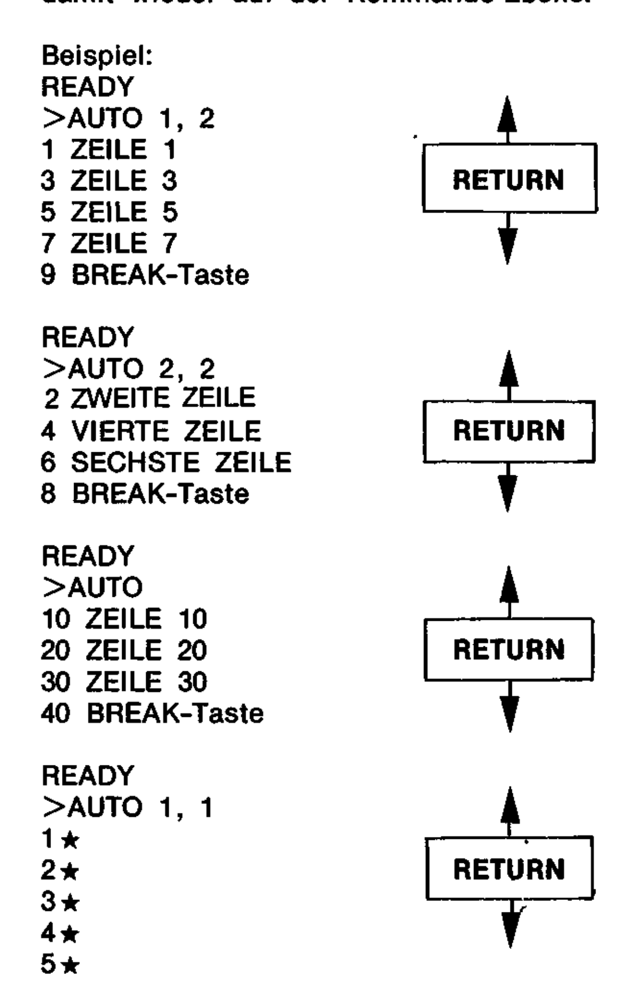
Ein Kommando wird mit der <RETURN>-Taste an den Computer abgeschickt, der es dann ausführt.

AUTO (Zeilennummer, Inkrement)

Dieses Kommando erzeugt automatisch Zeilennummern vor jeder Programmzeile, die Sie eingeben wollen. Die Optionen erlauben Ihnen, die Anfangszeilennummer und die Schrittweite zwischen den Zeilen anzugeben. Wird nun AUTO, wie bei allen Kommandos gefolgt von der <RETURN>-Taste, eingegeben, so nimmt der Computer als erste Zeilennummer die Zahl 10 und als Schrittweite auch die 10 an. Sie können den Programmbefehl direkt nach der Zeilennummer eingeben.

Beispiel:
30 PRINT "HIER IST ZEILE 30"

Nach jeder Eingabe einer Programmzeile erhöht der Computer automatisch die Zeilennummer um den im Inkrement angegebenen Betrag. Das AUTO-Kommando bleibt so lange aktiv, bis die BREAK-Taste gedrückt wird. Stößt der Computer im AUTO-Modus auf eine schon benutzte Zeile, so erscheint ein Stern „★" rechts neben der Zeilennummer. Wollen Sie diese Zeile nicht neu schreiben, so betätigen Sie die BREAK-Taste und sind damit wieder auf der Kommando-Ebene.


Beispiel:
READY
>AUTO 1, 2
1 ZEILE 1
3 ZEILE 3
5 ZEILE 5
7 ZEILE 7
9 BREAK-Taste

READY
>AUTO 2, 2
2 ZWEITE ZEILE
4 VIERTE ZEILE
6 SECHSTE ZEILE
8 BREAK-Taste

READY
>AUTO
10 ZEILE 10
20 ZEILE 20
30 ZEILE 30
40 BREAK-Taste

READY
>AUTO 1, 1
1★
2★
3★
4★
5★

CLEAR (Anzahl der Bytes)

Das CLEAR-Kommando löscht eine gewisse Zahl von Bytes und stellt sie zum Abspeichern von Zeichenketten zur Verfügung. Außerdem werden alle numerischen Variablen auf Null gesetzt und leere Zeichenketten in die Zeichenkettenvariablen geschrieben. CLEAR 100 ordnet den Zeichenkettenvariablen 100 Bytes zu. Wenn die Fehlermeldung „OS" (out of stringspace) auftritt, so kann mit dem CLEAR-Kommando mehr Platz für die Zeichenkettenvariablen freigegeben werden.

CONT

Wird ein Programm durch die BREAK-Taste unterbrochen oder hat das Programm einen STOP-Befehl ausgeführt, so kann die Ausführung des Programms mit CONT an der Stelle, an der es vorher unterbrochen wurde, wieder aufgenommen werden.

DELETE Zeilennummer (— Zeilennummer)

Dieses Kommando löscht die angegebene(n) Zeile(n).

Beispiele:

DELETE 5	Löscht Zeile 5
DELETE 7 — 10	Löscht Zeile 7, 10 und alle dazwischen
DELETE — 12	Löscht alle Zeilen vom Programmanfang bis einschließlich Zeile 12
DELETE .	Löscht die gerade eingegebene oder editierte Zeile

EDIT-Zeilennummer

Dieses Kommando veranlaßt den Computer, von der aktiven Kommandoebene in den Editierungsmodus überzugehen. Die Editierungsebene erlaubt es, Zeilen im Speicher zu verändern, ohne sie neu eintippen zu müssen. Das EDIT-Kommando hat eine Reihe von Subkommandos, die auf der Editierungsebene ausgeführt werden können. (Siehe Kapitel 21)Dem EDIT-Kommando muß eine gültige Zeilennummer folgen.

Beispiel: EDIT 20

Schaltet um auf die Editierungsebene, um Zeile 20 zu untersuchen.

LIST (Zeilennummer — Zeilennummer)

Mit diesem Kommando veranlassen Sie den Computer, eine oder mehrere Programmzeilen auf dem Schirm darzustellen. LIST ohne die Optionen in Klammern schreibt das gesamte Programm, daß sich momentan im Speicher befindet, heraus. Um den schnellen Durchlauf der Programmzeilen auf dem Schirm anzuhalten, betätigen Sie die SHIFT- und die @ -Taste gleichzeitig. Das Durchlaufen der Textzeilen (engl. up scrolling) wird durch Drücken einer beliebigen Taste wieder gestartet.

Beispiele:

LIST 3	Zeile 3 wird ausgegeben
LIST 10 — 20	Zeile 10, 20 und alle dazwischen
LIST — 50	von der ersten Zeile bis Zeile 50
LIST 20 —	von der Zeile 20 bis zum Programmende
LIST .	die Zeile, die gerade editiert wurde
LIST	alle Zeilen im Speicher

NEW

Dieses Kommando löscht alle Programmzeilen, setzt alle numerischen Variablen auf Null und schreibt in alle Zeichenkettenvariablen die leere Zeichenkette. Der Speicherplatz für die Zeichenkettenvariablen, der mit CLEAR gesetzt wurde, wird nicht verändert.

RUN (Zeilennummer)

Dieses Kommando startet ein Benutzerprogramm im Speicher. Wird die Zeilennummer nicht angegeben, so wird das Programm in der ersten Zeile gestartet. Wird jedoch eine Zeilennummer angegeben, so wird das Programm in dieser Zeile gestartet. Eingabe einer ungültigen Zeilennummer führt zu einer Fehlermeldung.

Mit jedem RUN-Kommando wird automatisch auch ein CLEAR-Kommando ausgeführt.

Beispiele:

RUN 50	Starte das Programm in Zeile 50
RUN	Starte das Programm in der ersten Zeile

SYSTEM

Dieses Kommando bringt den Computer in den Monitor-Modus. In diesem Modus können Maschinenprogramme von der Kassette geladen und ausgeführt werden. Nach Eingabe des SYSTEM-Kommandos erscheint ein „★?“ auf dem Schirm. Der Computer erwartet dann die Eingabe des Dateinamens der Maschinencodedatei (Objekt code), die er von der Kassette laden soll. Ist das Maschinenprogramm vollständig geladen, so erscheint ein weiteres „★?“. Um das Programm nun zu starten, tippen Sie einen /, gefolgt von der Startadresse des Maschinenprogramms (in dezimaler Schreibweise) ein. Wird nur ein / eingegeben, so startet das Programm an der Adresse, die in der Datei spezifiziert wurde.

Beim Laden erscheinen wie bei CLOAD 2 Sternchen — wird das linke zu einem „C“, liegt ein Ladefehler (CHECKSUM ERROR) vor.

TROFF

Dieses Kommando schaltet die TRON-Funktion aus. Normalerweise folgt es hinter einem TRON.

TRON

Dieses Kommando schaltet die Trace-Funktion ein. Sie erlaubt es, den Ablauf eines Programms zur Fehlersuche zu verfolgen. Sobald der Computer eine neue Programmzeile ausführt, wird ihre Zeilennummer in Klammern ausgegeben.

Beispiel:

```
10 PRINT"★★ PROGRAMM 1 ★★"
20 A = 1
30 IF A = 3 THEN 70
40 PRINT A
50 A = A + 1
60 GOTO 30
70 PRINT"★★ENDE VON PROGRAMM 1★★"
80 END
```

Geben Sie ein:
```
>TRON
>RUN
(10) ★★ PROGRAMM 1★★
( 20 ) ( 30 ) ( 40 ) 1
( 50 ) ( 60 ) ( 30 ) 2 ( 70 ) ★★ ENDE VON PROGRAMM 1 ★★
( 80 )
```

Soll die Programmausführung vor dem Programmende kurz unterbrochen werden, so kann das durch gleichzeitiges Betätigen der SHIFT- und der @-Taste erreicht werden. Um fortzufahren betätigen Sie irgendeine Taste. TRON und TROFF können auch in einem Programm verwandt werden.

Beispiel:
```
 .
 .
 .
90 IF A = B THEN 160
100 TRON
110 A = B + C
120 TROFF
 .
 .
 .
```

Ist bei der Ausführung dieses Programmteils A nicht gleich B, werden Zeile 100 bis 120 ausgeführt. Die Trace-Option wird eingeschaltet und das Durchlaufen des Befehls in Zeile 110 wird angezeigt. Danach wird die Trace-Option durch TROFF wieder abgeschaltet. TRON und TROFF können nach der Fehlersuche wieder aus dem Programm entfernt werden.

LLIST

Listet ein Programm auf dem Drucker. Dieses Kommando arbeitet genau so, wie das LIST-Kommando.

23. BASIC Programm Statements

Wir wollen in diesem Kapitel die Programmbefehle (= Statements) unserer BASIC-Programmiersprache erläutern. Im ersten Teil setzen wir uns mit den Ein- und Ausgabebefehlen, die dem Computer zur Verfügung stehen, um mit der Außenwelt in Verbindung zu treten, auseinander. Besonders behandelt werden solche, die mit dem Bildschirm und der Tastatur arbeiten und die Datenspeicherung auf der Kassette ermöglichen.

Der zweite Teil dieses Kapitels handelt von verschiedenen Funktionen der Programmbefehle (= Statements), die vom Video Genie System interpretiert werden. Da dies eine recht große Anzahl von Statements ist und jedes seine eigenen, charakteristischen Eigenschaften hat, wird dem Benutzer nahe gelegt, jedes Statement mit Hilfe der angegebenen Beispiele genau zu studieren.

Ein-Ausgabestatements

PRINT Item Liste

Schreibt ein Item oder eine Itemliste auf den Schirm. Als Item wird folgendes aufgefaßt:

a) Numerische Konstante	(Zahlen wie 0, 36872, 0.2, —34)
b) Zeichenkonstanten	(Zeichen in Anführungszeichen, wie etwa „VIDEO GENIE SYSTEM", „123=$%" usw.)
c) Zeichenvariablen	(Namen, die eine Zeichenkonstante repräsentieren, wie A$, X$, PR$ usw.)
d) Ausdrücke	(Verknüpfungen der obigen Items, wie (X+10)/Y oder „COMP" + „UTER" usw.)

Die Items in der Itemliste können durch Kommas oder Semikolons getrennt werden. Werden Kommas benutzt, so geht der Cursor automatisch zur nächsten Schreibzone, bevor er das Item ausgibt. Werden Semikolons benutzt, so wird kein Platz zwischen den Items gelassen; eines wird an das andere gehängt. Bei numerischen Items wird allerdings eine Leerstelle gelassen.

Beispiel:

```
10 N = 25 + 7
20 PRINT "25 + 7 IST GLEICH" ; N
30 END

READY
>RUN

25 + 7 IST GLEICH 32
```

Beispiel:

```
10 H$ = "HEIM"
20 C$ = "COMPUTER"
30 PRINT "PROBIEREN SIE UNSEREN " ; H$ ; C$
40 END

READY
>RUN

PROBIEREN SIE UNSEREN HEIMCOMPUTER
```

Werden Kommas benutzt, um die Items zu trennen, so erzeugt der Computer 4 Spalten pro Zeile. Jede Spalte kann maximal 9 Zeichen enthalten. Zeichen, die darüber hinausgehen, werden in die nächste Zeile geschrieben.

Beispiel:

```
10 PRINT "SPALTE 1", "SPALTE 2", "SPALTE 3", "SPALTE 4",  "SPALTE 5"
20 END

READY
>RUN

SPALTE 1 SPALTE 2 SPALTE 3 SPALTE 4
SPALTE 5
```

Werden zwei oder mehr Kommas angegeben, so erzeugt jedes 9 Leerstellen (blanks).

Beispiel:

```
10 PRINT "SPALTE 1" , ,"SPALTE 2"
20 END

READY
>RUN

SPALTE 1                    SPALTE 2
```

Beachten Sie folgendes Beispiel:

```
10 PRINT "ZEILE 1"
20 PRINT "ZEILE 2"
30 END

READY
>RUN

ZEILE 1
ZEILE 2
```

```
10 PRINT "ZEILE 1",
20 PRINT "ZEILE 2"
30 END

READY
>RUN

ZEILE 1   ZEILE 2
```

PRINT @ Stelle, Item List

Dieses Statement gibt die Items der Itemliste an der angegebenen Stelle auf dem Schirm aus. Das „ @ "-Zeichen muß unmittelbar auf das PRINT folgen und Stelle darf Werte zwischen 0 und 959 annehmen.

Beispiel:

```
20 PRINT @ 100, "STELLE 100"
```

Wenn der Benutzer ein PRINT-Statement eingibt, das auf die letzte Zeile des Schirms schreibt, wird automatisch ein Zeilenvorschub erzeugt, der alle Zeilen um eine Position nach oben wandern läßt. Um dies zu unterbinden, setzen Sie ein Semikolon an das Ende des Statements.

Beispiel:

```
10 PRINT @ 920, "LETZTE ZEILE";
```

PRINT TAB (Ausdruck)

Erlaubt es, den Cursor an jede beliebige Stelle in der Zeile zu setzen. Man kann mehr als ein TAB in einem PRINT-Statement benutzen. Doch der Wert von Ausdruck muß zwischen 0 und einschl. 255 liegen.

Beispiel:

```
10 PRINT TAB( 10 ) "POSITION 10" TAB( 30 ) "POSITION 30"

READY
>RUN

          POSITION 10          POSITION 30

10 N = 4
20 PRINT TAB (N) "POS" ; N TAB (N + 10) "POS" ; N + 10 TAB (N + 20) "POS" ;
   N + 20
30 END

READY
>RUN

POS 4          POS 14          POS 24
```

PRINT USING Format, Item Liste

Dieses Statement erlaubt es, Daten in einem vorher festgelegten Format auszugeben. Daten können numerische oder Zeichenkettenwerte sein.

Die Format- und Item-Liste im PRINT USING-Statement kann Variablen oder Konstanten enthalten. Das Statement schreibt die Items in der Form, wie es in der Formatangabe vorgegeben wird.

Die folgenden Deskriptoren können in dem Formatfeld benutzt werden:

\# Dieses Zeichen repräsentiert die richtige Stellung jeder Dezimalstelle in einer Zahl der Itemliste. Die Anzahl der # Zeichen bildet das erwünschte Format. Ist das Formatfeld größer als die Anzahl der Stellen in der Zahl, so werden die unbenutzten Feldpositionen links von der Zahl als Leerzeichen (blanks) und die rechts vom Dezimalpunkt als Nullen ausgegeben. Der Dezimalpunkt kann irgendwo in das durch die # Zeichen gebildete Formatfeld gesetzt werden. Werden Nachkommastellen unterdrückt, so wird gerundet. Wird ein Komma in eine Position zwischen der ersten Ziffer und dem Dezimalpunkt gesetzt, so erscheint in der Ausgabe nach je drei Vorkommaziffern ein Komma (beachte: der Computer benutzt die angloamerikanische Dezimalschreibweise, das deutsche Dezimalkomma entspricht hier dem Punkt).

Beispiel:

```
10 INPUT "FORMAT EINGEBEN" ; F$
20 IF F$ = "STOP" END
30 INPUT "ZAHL EINGEBEN" ; N
40 PRINT USING  F$ ; N
```

Dieses Programm fragt nach einer Formatliste und nach einem Item (in diesem Fall ein numerischer Wert). Es stoppt erst, wenn „STOP“ eingegeben wird.

Nun probieren Sie das Programm aus:

```
READY
>RUN

FORMAT EINGEBEN ? # # . # #
ZAHL EINGEBEN ? 12.34
12.34
FORMAT EINGEBEN ? # # # . # #
ZAHL EINGEBEN ? 12.34
  12.34
FORMAT EINGEBEN ? # # . # #
ZAHL EINGEBEN ? 123.45
%123.45
FORMAT EINGEBEN ? STOP
```

Das %-Zeichen wird ausgegeben, wenn das angegebene Feld für die Zahl zu klein ist. Die letzte Zahl links vom Dezimalpunkt wird nach dem %-Zeichen ausgegeben.

Nun starten wir das obige Programm wieder:

```
READY
>RUN

FORMAT EINGEBEN ? # # . # #
ZAHL EINGEBEN ? 12.345
12.35
FORMAT EINGEBEN ? STOP
```

Weil im Format nur zwei Nachkomma (bzw. Punkt)-Stellen angegeben sind, wird aufgerundet.

★★ Zwei Sternchen am Anfang des Formatfeldes bewirken, daß die unbenutzten Positionen links vom Dezimalpunkt mit Sternchen aufgefüllt werden. Die zwei Sternchen erzeugen zwei Feldpositionen mehr.

$$ Zwei Dollarzeichen am Anfang des Formatfeldes erzeugen ein gleitendes Dollarzeichen. D.h. ein Dollarzeichen wird vor der höchsten Stelle der Zahl ausgegeben.

★★$ Kombiniert den Effekt von ★★ und $$. Jede leere Position links von der Zahl wird mit Sternchen aufgefüllt und das Dollarzeichen wird vor der höchsten Stelle der Zahl ausgegeben.

Betrachten Sie das Beispielprogramm von vorhin:

```
READY
>RUN

FORMAT EINGEBEN ? ★★ ## . ##
ZAHL EINGEBEN ? 12.3
★★ 12.3
FORMAT EINGEBEN ? $$ ## . ##
ZAHL EINGEBEN ? 12.34
$12.34
FORMAT EINGEBEN ? ★★$ ### . ##
ZAHL EINGEBEN ? 12.34
★★★$12.34
FORMAT EINGEBEN ? STOP
```

\+ Wird ein + Zeichen am Anfang oder am Ende des Formatfeldes angegeben, so schreibt der Computer ein + Zeichen für positive Zahlen und ein — Zeichen für negative Zahlen am Anfang bzw. Ende der Zahl.

— Wird ein — Zeichen an das Ende des Formatfeldes gesetzt, so wird ein — Zeichen nach jeder negativen Zahl ausgegeben und ein Leerzeichen für positive Zahlen.

Beispiel:

```
READY
>RUN

FORMAT EINGEBEN? #### , #
ZAHL EINGEBEN ? 12345.6
12,346

FORMAT EINGEBEN ? + ## . ##
ZAHL EINGEBEN ? — 12.34
— 12.34
FORMAT EINGEBEN ?## . ## +
ZAHL EINGEBEN ? —12.34
12.34—
FORMAT EINGEBEN ? ## . ##
ZAHL EINGEBEN ? 12.34
12.34
FORMAT EINGEBEN ? ## . ###
ZAHL EINGEBEN ? 123456
%123456.000
FORMAT EINGEBEN ? STOP
```

%LEERZEICHEN%
Dient zum Definieren eines Zeichenfeldes, das mehr als ein Zeichen enthält. Die Länge des so formatierten Feldes ist die Zahl der Leerzeichen zwischen den Prozentzeichen plus 2. Ein Ausrufezeichen bringt den Computer dazu, nur das erste Zeichen einer aktuellen Zeichenkette zu benutzen.

Beispielprogramm:

```
10 INPUT "FORMAT EINGEBEN" ; F$
20 IF F$ = "STOP" END
30 INPUT "ZEICHENKETTE EINGEBEN" ; C$
40 PRINT USING F$ ; C$
50 GOTO 10
```

Dieses Programm arbeitet ähnlich wie das vorherige, nur wird jetzt keine numerische Variable mehr benutzt, sondern eine Zeichenkettenvariable.

Nun starten wir das Programm:

```
READY
>RUN

FORMAT EINGEBEN ? !
ZEICHENKETTE EINGEBEN ? ABCDE
A
FORMAT EINGEBEN ?%%
ZEICHENKETTE EINGEBEN ? ABCDE

FORMAT EINGEBEN ?          %
ZEICHENKETTE EINGEBEN ? ABCDEF
ABCDE
FORMAT EINGEBEN ? STOP
```

! Mit dem Ausrufezeichen können Sie Zeichenketten hintereinander hängen.

Beispiel:

```
10 INPUT "GIB DREI ZEICHENKETTEN EIN" ; A$,B$,C$
20 PRINT "DAS ERGEBNIS IST:";:PRINT USING "!" ; A$ ; B$ ; C$
30 END

READY
>RUN

GIB DREI ZEICHENKETTEN EIN ? ABC, XYZ, IJK
DAS ERGEBNIS IST : AXI

GIB DREI ZEICHENKETTEN EIN ? A, COMPUTER, PROGRAMM
DAS ERGEBNIS IST : ACP
```

Benutzt man mehr als das eine Ausrufezeichen, wird der erste Buchstabe der Zeichenkette mit so vielen Leerzeichen dahinter ausgedruckt, wie Leerzeichen zwischen den Ausrufezeichen eingesetzt wurden.

Folgendes Beispiel:

```
10 INPUT "GIB DREI ZEICHENKETTEN EIN" ; A$,B$,C$
20 PRINT "DAS ERGEBNIS IST : ";: PRINT USING "! ! !" ; A$ ; B$ ; C$
30 END

READY
>RUN

GIB DREI ZEICHENKETTEN EIN? XYZ, FGH, ABC
DAS ERGEBNIS IST : X F A

GIB DREI ZEICHENKETTEN EIN ? A, COMPUTER, PROGRAMM
DAS ERGEBNIS IST: A C P
```

INPUT Itemliste

Dieses Statement veranlaßt den Computer, die Programmausführung zu unterbrechen und zu warten, bis der Benutzer Daten eines spezifizierten Typs und einer spezifizierten Anzahl auf der Tastatur eingegeben hat. Einzugebende Daten können numerisch oder Zeichenketten, abhängig vom Typ der Variablen, sein. Jedes Item (wenn mehr als eins eingegeben wird) muß durch ein Komma vom anderen getrennt werden.

Beispiel:

```
10 INPUT A$, B$, A, B
```

Bei Ausführung dieses Statements muß der Benutzer zwei Zeichenketten und zwei numerische Werte eingeben. Die Reihenfolge der Eingaben muß konsistent sein. Führt der Computer ein INPUT Statement aus, so schickt er ein Signal zum Bildschirm:

? ■

Und wartet auf die Eingabe(n). Man kann alle vier Werte in einer Zeile eingeben (durch Kommas getrennt).

In diesem Fall könnte die Eingabe wie folgt aussehen:

BANANE , APFEL , 59, 3.14 <RETURN>-Taste

Der Computer ordnet die Werte dann wie folgt zu:

A$ = "BANANE"
B$ = "APFEL"
A = 59
B = 3.14

Eine andere Methode, diese Daten einzugeben, wäre es, sie in separate Zeilen aufzuteilen. Nach jeder Betätigung der <RETURN>-Taste erinnert Sie der Computer dann daran, das er noch Daten erwartet, indem er ausgibt:

```
?? ■
```

bis er Werte für alle Variablen erhalten hat. Dann geht er zum nächsten Statement über. Die Eingaben müssen mit den Variablentypen kompatibel sein. Man darf also für eine numerische Variable keine Zeichenkette eingeben. Wenn solches versucht wird, reagiert der Computer mit:

```
? REDO

? ■
```

und erwartet neue Daten für alle Variablen der Itemliste.

Beispiel:

```
10 INPUT A$, A
20 PRINT A$, A
30 END

READY
>RUN

?STRING, 10
STRING        10

READY
>RUN


?DIES IST EINE ZEICHENKETTE, 13.5
DIES IST EINE ZEICHENKETTE        13.5

READY
>RUN

?ABCD, IJK
?REDO
?ABCDE
?? 25
ABCDE 25
```

Besteht eine Eingabezeichenkette nur aus Leerzeichen, so muß sie in Anführungszeichen eingeschlossen werden.

Um klarer zu machen, welche Eingabe nun erwartet wird, kann man jedem INPUT-Statement eine Nachricht mitgeben, die ausgedruckt wird, bevor der Computer mit dem ?-Zeichen Daten erwartet. Diese Nachricht muß dem INPUT-Statement unmittelbar folgen, muß in Anführungszeichen eingeschlossen sein und von einem Semikolon gefolgt werden.

Beispiel:

```
10 INPUT "ART DER WARE UND STÜCKZAHL" ; N$,S

READY
>RUN

ART DER WARE UND STÜCKZAHL ? ■
```

DATA Itemliste

Dieses Statement erlaubt es dem Benutzer, Daten im Programm anzugeben und mit dem READ-Statement auf sie zuzugreifen. Auf die Daten greift der Computer sequentiell, beginnend mit dem ersten Item und endend mit dem letzten, zu. Jedes Item darf eine Zeichenkette oder auch eine numerische Variable sein. Genau wie bei der Eingabe über die Tastatur, muß jede Zeichenkette, die entweder Leerzeichen oder Semikolons oder Kommas enthält, in Anführungszeichen eingeschlossen werden.

Die Anordnung der Items im DATA-Statement muß mit dem Typ der Variablen im dazugehörigen READ-Statement übereinstimmen (es darf nicht versucht werden, Zeichenketten in numerische Variablen zu lesen). Das DATA-Statement darf an beliebiger Stelle im Programm stehen.

Beispiel:

```
10 READ A$, B$, C, D
20 PRINT A$, B$, C, D
30 DATA"ZEICHEN", "EIN LANGER SATZ"
40 DATA 20,137.54
50 END

READY
>RUN
ZEICHEN   EIN LANGER SATZ    20
   137.54
```

READ Itemliste

Dieses Statement erlaubt es, Daten vom DATA-Statement zu lesen und Variablen zuzuordnen. Die Werte im DATA-Statement werden sequentiell vom READ-Statement gelesen. Sind alle Daten im ersten DATA-Statement gelesen, so liest das nächste vorkommende READ-Statement aus dem nächsten DATA-Statement. Sind dann alle Werte in allen DATA-Statements einmal gelesen und versucht danach ein READ-Statement,weitere Werte zu bekommen, so tritt ein „keine Daten mehr"-Fehler auf (out of data error) mit dem Code OD.

Beispiel:

```
10 READ C$
20 IF C$="EOF" GOTO 60
30 READ Q
40 PRINT C$,, Q
50 GOTO 10
60 PRINT : PRINT "ENDE DER LISTE" : END
70 DATA BUECHER, 4, BLEISTIFTE, 5, KUGELSCHREIBER, 6
80 DATA FUELLER, 6, TIPPEX, 7, EOF
READY
>RUN

BUECHER          4
BLEISTIFTE       5
KUGELSCHREIBER   6
FUELLER          6
TIPPEX           7

ENDE DER LISTE
```

RESTORE

Dieses Statement erlaubt es dem nächsten READ-Statement, das erste Item der ersten DATA-Liste zu lesen, auch wenn zuvor schon andere READS ausgeführt wurden.

Beispiel:

```
10 READ A$, A
20 PRINT A$, A
30 RESTORE
40 READ B$, B
50 PRINT B$, B
60 DATA "JUPP SCHMITT", 25, "FRANZ SCHULZ", 32, "DRACULA", 234
70 END


READY
>RUN

JUPP SCHMITT        25
JUPP SCHMITT        25
```

PRINT #-1, Itemliste

Dieses Statement gibt die Werte der angegebenen Variablen in Itemliste auf dem Kassettenrecorder aus. Der Recorder muß geeignet vorbereitet sein, bevor dieses Statement ausgeführt wird. Nähere Informationen darüber finden Sie im Kapitel 8.

Beispiel:

```
10 A$ = "ANFANG BAND"
20 B$ = 3.1416
30 C$ = 50
40 D$ = "DATEN"
50 PRINT #-1, A$, B$, C$, D$, "ENDE DER DATEI"
```

Dieses Programm weist zunächst den Variablen A$, B$, C$ und D$ Werte zu und schreibt sie dann auf Band. Beachten Sie, daß die Zeichenkettenkonstante „ENDE DER DATEI" genauso auf Band geschrieben wird, wie die Variablen. Sind die Daten einmal auf Band gespeichert, so können sie zurückgelesen werden. Das ist im Prinzip der gleiche Vorgang, wie beim Abspielen einer Musikkassette.

WICHTIG

Die gesamte Anzahl der einzelnen Zeichen in allen Variablen und Konstanten der Itemliste eines PRINT # Statements darf 247 nicht übersteigen. Werden mehr als 247 Zeichen angegeben, so werden die restlichen ignoriert. (Es zählen auch die Kommas, die der Trennung einzelner Items dienen.)

Beispiel:

```
10 PRINT #-1, A$, B$, C$, D$, E$
```

Nehmen wir an, die gesamte Anzahl der Zeichen in A$, B$, C$ und D$ beträgt 243 und E$ habe die Länge 35. E$ wird nicht auf Band gespeichert, der Versuch, E$ dann später wieder einzulesen, erzeugt einen „keine Daten mehr" Fehler. (out of data error: Code: OD)

Sollen numerische Variablen auf Band geschrieben werden, so sind sie mit STR$ vorher in Zeichenketten umzuwandeln (s. dort).

INPUT #-1, Itemliste

Dies Statement veranlaßt den Computer, Werte vom Rekorder zu lesen und sie in die Variablen der Itemliste zu schreiben.

Beispiel:

```
10 INPUT #-1, A$, B$, C$, D$
```

Dieses Statement liest Daten vom Rekorder. Der erste Wert wird A$ zugewiesen, der zweite B$ usw. Die PLAY-Taste beim Rekorder muß gedrückt sein.

Eine „keine Daten mehr" (out of Data, Code: OD) Fehlermeldung tritt auf, wenn nicht genug Daten für alle Variablen in der Itemliste auf Band stehen.

Für PRINT # umgewandelte Zahlen können mit VAL wieder in solche zurückgewandelt werden.

24. PROGRAMM-BEFEHLE

DEFINT Buchstabenbereich

Variablen, die mit einem Buchstaben aus dem definierten Buchstabenbereich beginnen, werden als ganze Zahlen (integers) behandelt und abgespeichert. Eine Typenangabe (mit %, $ usw.) aber überschreibt diese Typendefinition. Erklärt man eine Variable zur ganzzahligen Variable, so spart man damit nicht nur Speicherplatz sondern auch Computerzeit. Berechnungen mit ganzzahligen Variablen sind schneller als solche mit Variablen einfacher und doppelter Genauigkeit. Beachten Sie, daß eine ganzzahlige Variable nur Werte zwischen —32768 und +32767 annehmen kann.

Beispiel:

```
10 DEFINT X, Y, Z
```

Hat der Computer Zeile 10 ausgeführt, so werden alle Variablen, die mit X, Y oder Z anfangen, als ganzzahlig behandelt. Daher sind von da an X2, X3, YA, YB, ZI, ZJ ganzzahlige Variablen. Jedoch X1#, YA#, YB#, ZI#, ZJ# bleiben Variablen doppelter Genauigkeit weil die Typenfestlegung mit „#" die mit DEFINT überschreibt.

Beispiel:

```
10 DEFINT A — D
```

Erklärt alle Variablen, die mit A, B, C oder D beginnen, zu ganzzahligen Variablen.

Beachten Sie, das DEFINT zwar an beliebiger Stelle im Programm eingesetzt werden kann, aber es die Bedeutung von Variablen ohne explizite Typendeklaration ($, #, %) recht unkontrolliert ändern (Sprungbefehle) kann. Deshalb sollte es normalerweise an den Anfang eines Programms gestellt werden.

DEFSNG Buchstabenbereich

Variablen, die mit einem Buchstaben aus ‚Buchstabenbereich' beginnen, werden als Variablen einfacher Genauigkeit behandelt. Explizite Typenfestlegung (mit #, $, %) überschreibt diese Definition.

Variablen und Konstanten einfacher Genauigkeit werden mit 7 Stellen gespeichert und mit 6 Stellen ausgegeben. Alle numerischen Variablen haben einfache Genauigkeit, wenn nichts anderes angegeben wurde. Das DEFSNG-Statement dient vor allem dazu, Variablen neu zu definieren, die vorher als Variablen doppelter Genauigkeit oder ganzzahlige Variablen festgelegt wurden.

Beispiel:

```
10 DEFSNG A—D, Y
```

Macht alle Variablen, die mit A bis D oder Y beginnen, zu Variablen einfacher Genauigkeit. Aber A# bleibt weiterhin eine Variable doppelter Genauigkeit und Y% bleibt eine ganzzahlige Variable.

DEFDBL Buchstabenbereich

Variablen, die mit einem Buchstaben aus ‚Buchstabenbereich' beginnen, werden als Variablen doppelter Genauigkeit behandelt und gespeichert. Eine explizite Typfestlegung (mit !, $, %) kann diese Definition überschreiben. Variablen doppelter Genauigkeit ermöglichen das Rechnen mit 17 Stellen, wovon 16 ausgegeben werden.

Beispiel:

10 DEFDBL M—P, G

Macht alle Variablen, die mit M bis P oder G beginnen, zu Variablen doppelter Genauigkeit. (M% oder G# bleiben unbeeinflußt).

DEFSTR Buchstabenbereich

Variablen, die mit einem Buchstaben aus ‚Buchstabenbereich' beginnen, werden als Zeichenkettenvariablen behandelt und abgespeichert.

Explizite Typenfestlegung (mit #, !, %) überschreibt diese Definition. Wenn genug Platz für Zeichenketten zur Verfügung steht (siehe CLEAR), kann eine Zeichenkettenvariable bis zu 255 Zeichen aufnehmen.

Beispiel:

10 DEFSTR A—D

Macht alle Variablen, die mit einem Buchstaben von A bis D beginnen, zu Zeichenkettenvariablen, außer es wurde eine explizite Typfestlegung angegeben (#, !, %). Nach der Ausführung von Zeile 10, wird folgender Ausdruck richtig:

B3 = "EINE ZEICHENKETTE".

CLEAR n

Dieses Statement setzt alle Variablen auf Null. Wird eine Zahl n angegeben, so stellt der Computer n Bytes zur Speicherung von Zeichenketten ab.

Wird das Video Genie System eingeschaltet, so werden jedes Mal automatisch 50 Bytes zur Speicherung von Zeichenketten freigesetzt.

Das CLEAR-Statement wird dann kritisch, wenn der Computer während der Programmausführung auf einen „kein Platz für Zeichenketten" (out of stringspace, Code OS) Fehler stößt. Dieser Fehler tritt auf, wenn die Zeichenketten des Programms mehr Platz beanspruchen, als freigegeben wurde.

Beispiel:

10 CLEAR 1000

Setzt 1000 Bytes für Zeichenketten frei und löscht alle Variablen.

DIM Name (dim1,.....,dim n)

Dieses Statement legt eine Variable oder eine Liste von Variablen als Feld (array) fest. Die Zahl der Elemente jeder Dimension können mit dim1, dim2 usw. angegeben werden. Wird DIM nicht ausgeführt, so wird angenommen, das jede Dimension 11 Elemente hat. Die Anzahl der möglichen Dimensionen ist nur durch den Speicherplatz begrenzt.

Beispiel:

```
10 DIM A( 5 ), B ( 3, 4 ), C (2, 3, 3 )
```

Dieses Statement definiert A als eindimensionales Feld (Vektor) mit 6 Elementen (0 bis 5), B als zweidimensionales Feld mit 20 Elementen (4 mal 5) und C als dreidimensionales Feld mit 48 Elementen (3 mal 4 mal 4).

DIM darf an beliebiger Stelle in das Programm gesetzt werden. Die Dimensionsangabe darf ganzzahlig oder ein Ausdruck sein.

Beispiel:

```
10 INPUT "ANZAHL DER PUNKTE", N
20 DIM A( N+2, 4 )
```

Die Anzahl der Elemente von A kann, abhängig von N, verändert werden.
Um ein Feld neu zu dimensionieren, muß vorher ein CLEAR-Statement ohne Argument n eingegeben werden, sonst wird eine Fehlermeldung ausgegeben.

Beispiel:

```
10 X( 2 )=13.6
20 PRINT "DAS ZWEITE ELEMENT IST : " ; X( 2 )
30 DIM X ( 15 )
40 PRINT X ( 2 )
50 END

READY
>RUN
DAS ZWEITE ELEMENT IST 13.6
? DD ERROR IN 30
```

Durch den Gebrauch von X(2) in 10, vor dem DIM in 30, wurde es implizit mit DIM X(10) dimensioniert.

LET Variable = Ausdruck

Dieses Statement ordnet einer Variablen einen Wert zu. Das Wort LET ist in einem solchen Zuordnungsstatement beim Video Genie System BASIC nicht unbedingt erforderlich, wurde aber in seinen Sprachumfang aufgenommen, um Kompatibilität mit anderen Systemen zu gewährleisten.

Beispiel:

```
10 LET A = 5.67
20 B% = 20
30 S$ = "ZEICHEN"
40 LET D% = D% + 1
50 PRINT A, B%, S$,D%
60 END

READY
>RUN

5.67          20          ZEICHEN          1
```

In den obigen Zuordnungen wird in die Variable rechts vom Gleichheitszeichen der Wert des Ausdrucks links vom Gleichheitszeichen geschrieben. All diese Statements sind korrekt.

END

Dieses Statement bewirkt die Beendigung der Programmausführung. Das End-Statement wird primär zur Beendigung von Programmen an einer anderen Stelle als am Ende ihres logischen Textes eingesetzt.

Beispiel:

```
 5 B = 3 : C = 14
10 A = C + B
20 GOSUB 70
30 D = X + Y
40 PRINT "DAS ERGEBNIS IST : " ;
50 PRINT A, D
60 END
70 X = 50
80 Y = A ★ X
90 RETURN

DAS ERGEBNIS IST : 17          900
```

Das END-Statement in Zeile 60 hält den Computer davon ab, Zeile 70 bis 90 auszuführen. Damit kann das Unterprogramm, das in Zeile 70 beginnt, nur von der Zeile 20 ausgeführt werden.

STOP

Dieses Statement ist eine große Hilfe bei der Fehlersuche in Programmen. Es setzt einen Unterbrechungspunkt (break point) in der Programmausführung und erlaubt dann, Werte zu verändern oder auszugeben. Führt der Computer einen STOP-Befehl aus, so gibt er die Nachricht BREAK IN Zeilennummer (Unterbrechung in Zeile ...) aus. Mit dem aktiven Kommando CONT können Sie die Programmausführung an dem Punkt, an dem vorher unterbrochen wurde, wieder aufnehmen.

Beispiel:

```
 5 INPUT B, C
10 A = B + C
20 STOP
30 X = ( A + D ) / 0.74
40 IF X < 0 GOTO 70
50 PRINT A, B, C
60 PRINT X
70 END

READY
>RUN

? 2, 4
BREAK IN 20
READY
>PRINT A
6
READY
>CONT

6          2          4
8.10811
```

Das Stop-Statement gab uns die Möglichkeit, den Wert von A zu betrachten, bevor Zeile 30 ausgeführt wurde.

GOTO Zeilennummer

Dieses Statement übergibt die Kontrolle über das Programm an die angegebene Zeile (Sprungbefehl). Wird es unabhängig benutzt, so wird ein unbedingter Sprung ausgeführt. Ein Bedingungsstatement kann vor das GOTO gesetzt werden, um einen bedingten Sprung zu erzeugen.

Beispiel:

```
10 A = 10
20 B = 45
30 C = A + B
40 C = C * 3.4
50 GOTO 100
60 .
70 .
80 .
90 .
100 PRINT "A =" ; A, "B =" ; B, "C =" ; C
110 END
```

```
READY
>RUN

A = 10            B = 45            C = 187
```

Wird Zeile 50 ausgeführt, so wird die Kontrolle an Zeile 100 übergeben.

Beispiel:

```
10 IF A = 2 GOTO 120
```

Wird Zeile 10 ausgeführt und A ist gleich 2, dann springt der Computer zur Zeile 120. Ist A nicht gleich 2, geht er zum nächsten Statement.

Man kann das GOTO auch auf der aktiven Kommandoebene als Alternative zum RUN benutzen. Mit diesem Vorgehen vermeidet man, daß die Variablen gelöscht werden.

GOSUB Zeilennummer

Übergibt die Kontrolle an die Zeile, in der das angegebene Unterprogramm beginnt. Führt der Computer dann in dem Unterprogramm ein RETURN-Statement aus, so kehrt er zu dem dem GOSUB folgenden Statement im (Haupt) Programm zurück. Wie beim GOTO darf auch beim GOSUB ein Bedienungsstatement vorausgehen, wie etwa:

```
IF A = B THEN GOSUB 100
```

Beispiel:

```
10 PRINT "HAUPTPROGRAMM"
20 GOSUB 50
30 PRINT "HAUPTPROGRAMM ENDE"
40 END
50 PRINT "UNTERPROGRAMM"
60 RETURN

READY
>RUN

HAUPTPROGRAMM
UNTERPROGRAMM
HAUPTPROGRAMM ENDE
```

RETURN

Dieses Statement beendet ein Unterprogramm und übergibt die Kontrolle an das Statement, das dem GOSUB folgt. Ein Fehler tritt auf, wenn für ein RETURN kein entsprechendes GOSUB vorhanden war.

ON n GOTO Zeilennummern -Liste

Dieses Statement erlaubt es, gleich mehrere Sprungziele anzugeben. Gesprungen wird in Abhängigkeit von n. Das generelle Format des ON n GOTO ist:

ON Ausdruck GOTO 1. Zeilennummer, 2. Zeilennummer...

Der Wert von Ausdruck muß zwischen 0 und 255 einschl. sein.

Wird ein ON — GOTO-Statement ausgeführt, so wird zunächst der ganzzahlige Anteil von Ausdruck berechnet (entspricht INT(Ausdruck)). Dieses Ergebnis sei n. Dann sucht der Computer das n te Element der Zeilennummern-Liste und springt zu dieser Zeilennummer. Ist n nun größer als die Anzahl der angegebenen Zeilennummern, so wird das, auf das ON — GOTO-Statement folgende Statement ausgeführt. Ist n kleiner als Null, so tritt ein Fehler auf. Die Zeilennummern-Liste kann eine beliebige Anzahl von Zeilennummern enthalten.

Beispiel:

```
10 INPUT "KOMMANDO EINGEBEN" ; C
20 ON C GOTO 100, 120, 130, 150, 130
30 PRINT "ENDE DES PROGRAMMS" : END
100 PRINT "HIER ZEILE 100" : GOTO 10
120 PRINT "HIER ZEILE 120" : GOTO 10
130 PRINT "HIER ZEILE 130" : GOTO 10
150 PRINT "HIER ZEILE 150" : GOTO 10
READY
>RUN
KOMMANDO EINGEBEN? 5
HIER ZEILE 130
KOMMANDO EINGEBEN? 4
HIER ZEILE 150
KOMMANDO EINGEBEN? 1
HIER ZEILE 100
KOMMANDO EINGEBEN? 2
HIER ZEILE 120
KOMMANDO EINGEBEN? 3
HIER ZEILE 130
KOMMANDO EINGEBEN? 0
ENDE DES PROGRAMMS
READY
>RUN
KOMMANDO EINGEBEN? 4
HIER ZEILE 150
KOMMANDO EINGEBEN? 6
ENDE DES PROGRAMMS
```

Das ON — GOTO-Statement ist eine elegante Methode, um daß gleiche zu erreichen, was folgende IF-GOTO-Statements bewirken:

```
10 IF C = 1 GOTO 100
20 IF C = 2 GOTO 120
30 IF C = 3 GOTO 130
40 IF C = 4 GOTO 150
50 IF C = 5 GOTO 130
60 IF C < 1 OR C > 5 GOTO 70 : REM GEHE ZUM NÄCHSTEN STATEMENT
```

ON n GOUSUB Zeilennummern-Liste

Arbeitet wie ON n GOTO, nur statt der Sprünge werden Unterprogramme aufgerufen.

Beispiel:

```
10 PRINT "★★FUNKTION DER UNTERPROGRAMME★★"
20 PRINT "    1. FUNKTION A"
30 PRINT "    2. FUNKTION B"
40 PRINT "    3. FUNKTION C"
50 INPUT "GIB 1, 2 ODER 3 EIN" ; N
60 ON N GOSUB 150, 100, 250
70 END
100 PRINT "HIER FUNKTION B" : RETURN
150 PRINT "HIER FUNKTION A" : RETURN
250 PRINT "HIER FUNKTION C" : RETURN
READY
>RUN
★★FUNKTION DER UNTERPROGRAMME★★
1. FUNKTION A
2. FUNKTION B
3. FUNKTION C
GIB 1, 2 ODER 3 EIN? 2
HIER IST FUNKTION B
READY
>RUN
★★FUNKTION DER UNTERPROGRAMME★★
1. FUNKTION A
2. FUNKTION B
3. FUNKTION C
GIB 1, 2 ODER 3 EIN? 1
HIER IST FUNKTION A
```

FOR Name = Ausdruck TO Ausdruck STEP Ausdruck
NEXT Name

Diese Statements bilden eine iterative Schleife. Alle Statements, die zwischen FOR und NEXT stehen, werden einige Male ausgeführt.
Das allgemeine Format ist:

```
FOR Zähler = Anfangswert TO Endwert STEP Inkrement
★
★
★
Statements
★
★
★
NEXT Zähler
```

Beim FOR-Statement können Anfangswert, Endwert und Inkrement (Wert, um den hochgezählt wird) Konstanten, Variablen oder Ausdrücke sein. Wird das FOR-Statement das erste Mal ausgeführt, so werden sie berechnet und gespeichert. Verändern sich diese Werte in der Schleife, so hat das keine Auswirkungen auf die Arbeitsweise der Schleife selbst. Aber der Wert des Zählers darf nicht verändert werden, sonst arbeitet die Schleife nicht korrekt.

Die FOR — NEXT-Schleife funktioniert wie folgt:
Wird das FOR-Statement zum ersten Mal ausgeführt, so wird der Zähler auf den Anfangswert gesetzt. Das Programm wird weiter ausgeführt, bis ein NEXT-Statement gefunden wird. Jetzt wird der Zähler um den Wert von Inkrement, der nach dem STEP angegeben wurde, erhöht. Wird STEP und Inkrement nicht angegeben, so wird automatisch für Inkrement eine 1 angenommen. Ist Inkrement negativ, so wird der Zähler heruntergezählt. Der Zähler wird dann mit dem Endwert des FOR-Statements verglichen. Ist der Zähler größer als der Endwert, so wird die Ausführung der Schleife eingestellt und die Programmausführung geht nach dem NEXT-Statement weiter. (Ist Inkrement negativ, so endet die Ausführung, wenn der Zähler kleiner als der Endwert ist). Hat der Zähler den Endwert noch nicht überschritten, so geht der Computer zurück zum ersten Statement nach dem FOR-Statement.

Beispiel:

```
10 FOR K = 0 TO 1 STEP 0.3
20 PRINT "WERT VON K : " ; K
30 NEXT K
40 END

READY
>RUN

DER WERT VON K : 0
DER WERT VON K : .3
DER WERT VON K : .6
DER WERT VON K : .9
```

Dann ist K = 1.2 und damit größer als der Endwert 1. Deshalb endet die Schleife dort und druckt 1.2 nicht mehr.

Beispiel:

```
10 FOR N = 5 TO 0
20 PRINT "DER WERT VON N : " ; N
30 NEXT N
40 END
READY
>RUN
DER WERT VON N : 5

10 FOR N = 5 TO 0 STEP —1
20 PRINT "DER WERT VON N : " ; N
30 NEXT N
40 END
READY
>RUN
DER WERT VON N : 5
DER WERT VON N : 4
DER WERT VON N : 3
DER WERT VON N : 2
DER WERT VON N : 1
DER WERT VON N : 0
```

Wird kein Step angegeben, so wird automatisch STEP 1 angenommen. N wird das erste Mal hochgezählt, wird zu N = 6 und ist damit größer als der Endwert 0; die Schleife endet. Der Vorgang verläuft anders, wenn STEP —1 angegeben wurde (s. o.)

Beispiel:

```
10 FOR A = 0 TO 3
20 PRINT "DER WERT VON A : " ; A
30 NEXT
40 END
READY
>RUN
DER WERT VON A : 0
DER WERT VON A : 1
DER WERT VON A : 2
DER WERT VON A : 3
```

Statt NEXT A können wir wie in Zeile 30, auch einfach NEXT schreiben. Bei der Programmierung von verschachtelten FOR NEXT-Schleifen ist es jedoch günstig, anzugeben, welche Schleife man nun mit dem konkreten NEXT abgeschlossen hat.

Hier ist ein Beispiel für geschachtelte Schleifen, daß zeigen soll, wie man den Zähler in jedem NEXT-Statement identifiziert:

```
10 I = 1
20 J = 2
30 K = 3
40 FOR N = I+1 TO J+1
50 PRINT "ERSTE SCHLEIFE"
FOR M = 1 TO K
70 PRINT "ZWEITE SCHLEIFE"
80 NEXT M
90 NEXT N
100 END

READY
>RUN

ERSTE SCHLEIFE
ZWEITE SCHLEIFE
ZWEITE SCHLEIFE
ZWEITE SCHLEIFE
ERSTE SCHLEIFE
ZWEITE SCHLEIFE
ZWEITE SCHLEIFE
ZWEITE SCHLEIFE
```

ERROR Code

Dieses Statement wird benutzt, um in eine ON ERROR GOTO Routine zu verzweigen. Führt der Computer ein ERROR-Code-Statement aus, so verhält er sich genau so, als wäre dieser Fehler aufgetreten. Er simuliert die Fehlersituation.

Beispiel:

```
30 ERROR 1

READY
>RUN

? NF ERROR IN 30
```

ON ERROR GOTO Zeilennummer

Dieses Statement erlaubt es, ein Abfangprogramm für Fehler (engl. error trapping routine) aufzurufen. Damit kann man, kommt es bei der Programmausführung zu einem Fehler, erreichen, daß das Programm nicht mit einer Fehlermeldung anhält, sondern in ein Programm verzweigt, das die Ursache des Fehlers erkennt und beseitigt. Meist hat der Benutzer einen bestimmten Fehlertyp im Auge, wenn er das ON ERROR GOTO Statement verwendet. Führt das Programm z. B. eine Division durch und der Fall der Division durch Null ist nicht von vornherein ausgeschlossen worden, so kann die verbotene Division durch Null mit einem Fehlerbehandlungs-Programm abgehandelt werden.

Beispiel:

```
5 B = 15 : C = 0
10 ON ERROR GOTO 120
20 A = B / C
30 PRINT A, B, C
40 END
120 PRINT "MAN DARF NICHT DURCH NULL TEILEN ! !"
130 RESUME 40

READY
>RUN

MAN DARF NICHT DURCH NULL TEILEN ! !
```

In diesem Beispiel hat C den Wert 0 und in Zeile 20 kommt es zu einer Division durch Null, was normalerweise eine Fehlermeldung und Beendigung des Programms zur Folge hätte. Wegen Zeile 10 ignoriert der Computer aber diese Situation einfach und geht in die Zeile 120, zur Fehlerbehandlungsroutine. Beachten Sie, daß das ON ERROR GOTO Statement vor dem möglichen Auftreten des Fehlers stehen muß, sonst hat es keine Wirkung. Außerdem muß die Fehlerbehandlungsroutine mit RESUME abgeschlossen werden.

RESUME Zeilennummer

Dieses Statement beendet eine Fehlerbehandlungsroutine und gibt an, wo die normale Programmausführung weiter gehen soll.

RESUME 0 oder RESUME ohne Zeilennummer bewirkt, daß der Computer zu dem Statement zurückkehrt, in dem es zu einem Fehler gekommen war. Ist eine Zeilennummer angegeben, so geht er zu dieser Zeile zurück.

RESUME NEXT veranlaßt den Computer, zu dem Befehl zurück zu gehen, der hinter dem Befehl steht, in dem der Fehler erkannt wurde.

Beispiel:

```
10 ON ERROR GOTO 80
20 PRINT "EINFACHE DIVISION."
30 INPUT "GIB ZWEI ZAHLEN EIN" ; A, B
40 IF A = 0 END
50 C = A / B
60 PRINT "DER QUOTIENT IST : " ; C
70 GOTO 20
80 PRINT "VERSUCH, DURCH NULL ZU TEILEN"
90 PRINT "VERSUCHEN SIE ES NOCH EINMAL..."
100 RESUME 20
```

```
READY
>RUN

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN? 6, 2
DER QUOTIENT IST : 3

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN? 7,3
DER QUOTIENT IST : 2.33333

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN? 5, 0
VERSUCH, DURCH NULL ZU TEILEN
VERSUCHEN SIE ES NOCH EINMAL...
EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN? 9, 4
DER QUOTIENT IST : 2.25

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN? 0, 0
```

REM

REM gibt Fußnoten an. Dieses Statement informiert den Computer, daß in dieser Zeile nur noch Kommentare folgen. Diese werden im Programmablauf ignoriert. Es ermöglicht dem Benutzer, durch die Kommentare übersichtlichere Programme zu schreiben.

Beispiel:

```
10 REM ★ BEDEUTUNG DER VARIABLEN ★
20 REM ★  A = AUFWAND             ★
20 REM ★  B = TEILEZAHL           ★
40 REM ★  C = KOSTEN DER EINHEIT  ★
50 REM -------------------------
60 A = B ★ C: REM AUFWAND = ZAHL DER TEILE MAL KOSTEN
```

IF Ausdruck Aktion

Dieses Statement bringt den Computer dazu, einen logischen oder relationalen Ausdruck zu entscheiden. Ist der Ausdruck „wahr", so wird Aktion ausgeführt, ist er falsch, so wird Aktion ignoriert und mit dem nächsten Statement fortgefahren. Numerisch bedeutet „wahr", daß der Wert von Ausdruck nicht Null ist.

Beispiel:

```
10 INPUT "GIB ZAHL EIN ( MAX. 20 ) " ; A
20 IF A > 20 GOTO 60
30 A = A ★ 3.1416 ★ 2
40 PRINT "DER KREISUMFANG IST : " ; A
50 END
60 PRINT "ZAHL ZU GROSS, MAX. 20" : GOTO 10

READY
>RUN

GIB ZAHL EIN ( MAX. 20 ) ? 24
ZAHL ZU GROSS, MAX. 20
GIB ZAHL EIN ? 18
DER KREISUMFANG IST : 113.098
```

In diesem Beispiel wird jedes Mal, wenn eine Zahl größer als 20 eingegeben wird, eine Warnung ausgegeben und eine neue Eingabe erwartet. Ist A jedoch kleiner oder gleich 20, ignoriert der Computer das GOTO 60 in Zeile 20 und beendet seine Berechnung ohne eine Warnung auszugeben.

Beispiel:

```
120 INPUT A : IF A = 10 AND A < B THEN 160
120 INPUT A : IF A = 10 AND A < B GOTO 160
```

Beide Statements haben die gleiche Wirkung.

THEN Statement oder Zeilennummer

Bezeichnet den Beginn der „Aktion“ in einem IF-THEN Statement. THEN kann entfallen, wenn es benutzt wird, um eine Zeilennummer als Sprungziel anzugeben, wie in: IF A = B GOTO 100. THEN sollte in F-THEN-ELSE-Statements angegeben werden.

ELSE Statement oder Zeilennummer

Dieses Statement kann nur hinter einem IF Statement stehen und gibt eine Alternative an, wenn der Ausdruck im IF Statement „falsch“ wird.

Beispiel:

```
10 IF A = 1 THEN 60 ELSE 40
```

In diesem Beispiel wird nach 60 verzweigt, wenn A gleich 1 ist, ist dies nicht der Fall, wird nach 40 verzweigt. IF-THEN-ELSE-Statements können verschachtelt werden, aber die Anzahl der IFs und ELSEs müssen übereinstimmen.

Beispiel:

```
10 INPUT "GIB DREI ZAHLEN EIN" ; X, Y, Z
20 PRINT "DIE GRÖSSTE ZAHL IST : " ;
30 IF X < Y OR X < Z THEN IF Y < Z THEN PRINT Z ELSE PRINT Y ELSE PRINTX
40 END

READY
>RUN

GIB DREI ZAHLEN EIN? 30, 75, 73
DIE GRÖSSTE ZAHL IST : 75
```

Das Programm erwartet drei Zahlen als Eingabe und gibt die größte der drei aus.

LPRINT

Gibt Daten auf dem Drucker aus. Dieses Statement (auch Kommando) arbeitet analog zum PRINT Statement. Ist kein Drucker angeschlossen, so gerät der Computer bei Ausführung dieses Statements in eine endlose Schleife, aus der er nur durch gleichzeitiges Drücken der beiden < RST >-Tasten befreit werden kann.

Beispiel:

```
10 FOR X = 1 TO 0 STEP —0.25
20 LPRINT "X BETRÄGT : " ; X
30 NEXT X
40 END

READY
>RUN
```

```
X BETRÄGT : 1
X BETRÄGT : .75
X BETRÄGT : .5
X BETRÄGT : .25
X BETRÄGT : .0
```

25. VERARBEITUNG VON FELDERN

Ein Feld (array) ist einfach eine geordnete Liste von Daten. Das Video-Genie-System verarbeitet numerische und auch Zeichenketten-Felder. Beide Datentypen können aber nicht im gleichen Feld gemischt auftreten. Die Konzepte der Programmierung von Feldern sind sehr wichtig für die Computeranwendung und deshalb sollte der Benutzer versuchen, die Beispiele dieses Kapitels zu verstehen.

Nehmen wir an, Fritz Walter studiert an einer Universität. Das Gebäude hat drei Stockwerke, von denen jedes vier Klassenräume beherbergt. Jeder Raum hat 45 Plätze. Fritz hat eine Vorlesung in Geschichte belegt und es sind nur 36 Studenten mit ihm in der Klasse. Sehen wir uns die Namensliste von Fritzs Klasse an:

NAMENLISTE

1. Maria Adam
2. Jupp Braun
3. Heinrich Cox

★
★
★

36. Fritz Walter

Um eine Person in der Liste zu finden, müssen wir die Liste lediglich von oben nach unten durchlesen. Die Methode, mit der der Name gesucht wird, ist nicht sonderlich wichtig.

Wichtig ist, wie man eine Person in der Liste finden kann, wenn man nur ihre Nummer kennt. In der obigen Liste ist Maria Adam die erste, Jupp Braun die zweite Person usw.... Die Zahlen geben also einen systematischen Weg an, um eine Person zu finden.

Benutzen wir den Computer, um die Namensliste aufzuzeichnen, können wir jedem Namen eine einheitliche Variable zuweisen:

```
10 N0$ = "MARIA ADAM"
20 N1$ = "JUPP BRAUN"
30 N2$ = "HEINRICH COX"
.
.
.
100 NS$ = "THOMAS HAGEN"
.
.
.
140 NZ$ = "FRITZ WALTER"
```

Bedenkt man, daß 36 Studenten in der Klasse sind, so sieht man, daß dies eine uneffektive und zeitraubende Methode ist. Sehen wir das Programm genauer an, so stellen wir fest, daß alle Variablen laufende Nummern angehängt bekommen haben. Das haben wir einfach intuitiv und von Hand gemacht. Erzeugt der Computer nun diese laufenden Nummern selbst, so ist das ein Feld (array). Wir definieren ein Feld AR$ mit 45 Elementen (45 Plätze) und ordnen jedem Element einen Namen zu.

Beispiel:

```
5 CLEAR 1000 : REM SETZE 1000 BYTES FÜR ZEICHENKETTEN FREI
10 DIM AR$ ( 44 ) : REM FELD AR$ HAT 45 ELEMENTE
20 FOR N = 0 TO 44 : REM 45 SCHLEIFENDURCHLÄUFE
30 INPUT "GIB DEN NAMEN DES STUDENTEN EIN" ; AR$ ( N )
40 REM ORDNE DEN FELDELEMENTEN DIE NAMEN ZU
50 NEXT N
60 END
```

Dieses Programm liest 45 Namen ein und speichert sie in dem Feld AR$. Nach der Programmausführung haben wir folgende Variablenbelegung:

Das Element AR$ (0) hat den Wert "Maria Adam"
Das Element AR$ (1) hat den Wert "Jupp Braun"
Das Element AR$ (2) hat den Wert "Heinrich Cox"

Das Element AR$ (35) hat den Wert "Fritz Walter"

Wollen wir die Liste nun ausdrucken, können wir folgendes Programm einsetzen:

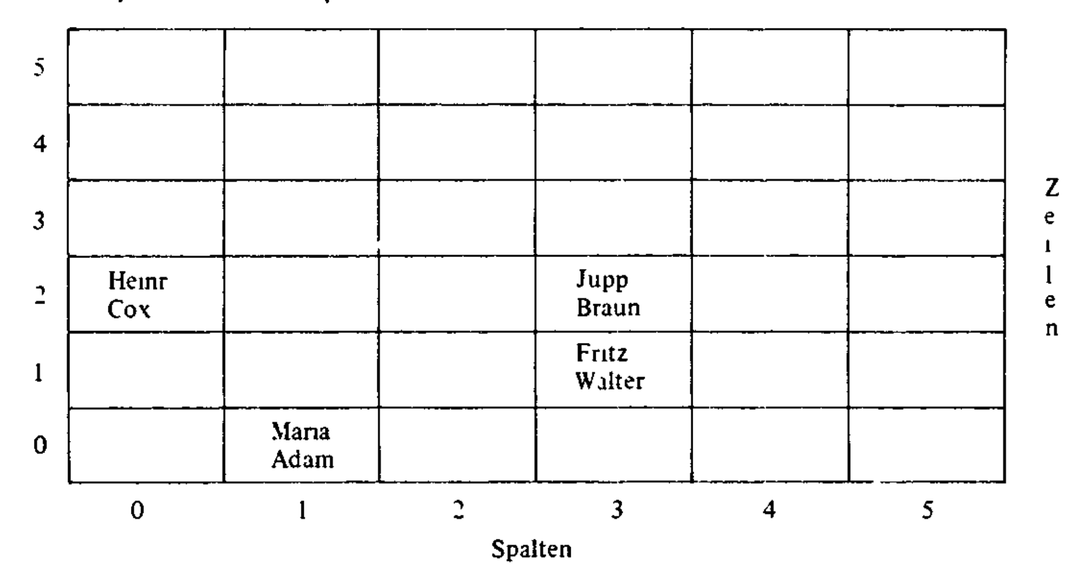
```
5 CLEAR 1000 : REM SETZE 1000 BYTES FÜR ZEICHENKETTEN FREI
10 DIM AR$ ( 44 ) : REM FELD AR$ SOLL 45 ELEMENTE BEKOMMEN
20 REM ★★EINGABE DER FELDELEMENTE ★★
20 FOR N = 0 TO 44 : REM 45 SCHLEIFENDURCHLÄUFE
30 INPUT "GIB DEN NAMEN DES STUDENTEN EIN" ; AR$ ( N )
40 REM ORDNE DEN FELDELEMENTEN DIE NAMEN ZU
50 NEXT N
60 REM ★★ AUSDRUCKEN DER FELDELEMENTE ★★
70 FOR N = 0 TO 44 : REM WIEDER DIE 45 MAL SCHLEIFE
80 PRINT AR$ ( N ) : REM DRUCKE DIE FELDELEMENTE ( NAMEN )
90 NEXT N
100 END
```

Der Ausgabeteil des Programms ersetzt folgende 45 PRINT-Statements:

```
10 PRINT N0$
20 PRINT N1$
30 PRINT N2$
40 .
50 .
60 .
70 PRINT NS$
80 .
90 .
100 .
110 PRINT NZ$
```

Nun sollten Sie ein Gefühl für die Nützlichkeit der Felder bekommen haben.

Angenommen, ein Lehrer dieser Klasse möchte nun einen Sitzplan aufstellen. Es sind 6 Zeilen und 6 Spalten von Sitzplätzen vorhanden:



Die Sitzpositionen unserer vier Beispiel-Studenten sind im obigen Plan eingezeichnet. Da sie sich aber nicht der Namensliste folgend hingesetzt haben, müssen wir uns eine andere Methode ausdenken, um auf den Sitzplan zuzugreifen. Will der Lehrer z. B. feststellen, ob Fritz Walter abwesend ist, so muß er nachsehen, ob der Stuhl in Zeile 1, Spalte 3 leer ist oder nicht. Er kann auch in Zeile 2 Spalte 0 nach Heinrich Cox suchen. Der Computer tut nun genau das gleiche, wie dieser Lehrer. Wir können diesen Sitzplan in ein 2 dimensionales Feld SP$ (5, 5) abbilden. Die erste 5 steht für die Zeilen, die zweite für die Spalten. Wollen wir nun Jupp Braun aufrufen, so müssen wir in SP$ (2, 3) nachschauen.

Nun wollen wir unseren Sitzplan ausdrucken. Das leistet folgendes Programm:

```
10 CLEAR 1000 : DIM SP$ ( 5, 5 ) : REM SP$ IST 6X6 FELD
20 FOR R = 5 TO 0 STEP —1
30 REM ZEILE 5 BIS ZEILE 0 AUSDRUCKEN
40 FOR C = 0 TO 5
50 REM SCHLEIFE, UM DIE NAMEN ALLER SPALTEN AUSZUGEBEN
60 PRINT SP$ ( R , C );: REM DRUCKE NAMEN IN ZEILE R UND SPALTE C
70 NEXT C
80 PRINT : REM PRINT ALLEIN = ZEILENVORSCHUB
90 NEXT R
100 END
```

Dieses Programm druckt den Sitzplan in Tabellenform. Es beginnt bei der letzten Sitzreihe der Klasse und endet mit der ersten. Das Programm initialisiert zuerst R = 5 und C = 0. Dann druckt es die Werte der Elemente.

SP$ (5, 0) ; SP$ (5 , 1) ; SP$ (5, 2) ; SP$ (5, 3).....SP$ (5, 5)

An diesem Punkt wird der Wert von C = 5. Der Computer springt von der inneren „C"-Schleife zurück in die äußere „R"-Schleife. Subtrahiert 1 von R und R wird damit zu 4. C wird wieder zu Null zurückgesetzt und der Vorgang beginnt mit R = 4 neu:

SP$ (4, 0); SP$ (4, 1) ; SP$ (4, 2) ; SP$ (4, 3).....SP$ (4, /)

Der gesamte Prozeß wird wiederholt, bis R = -1. Dann stoppt das Programm. Die endgültige Liste sieht so aus:

SP$ (5, 0) ; SP$ (5, 1) ; SP$ (5, 2).... ..SP$ (5, 5)
SP$ (4, 0) ; SP$ (4, 1) ; SP$ (4, 2).... ..SP$ (4, 5)
SP$ (3, 0) ; SP$ (3, 1) ; SP$ (3, 2).... ..SP$ (3, 5)
SP$ (2, 0) ; SP$ (2, 1) ; SP$ (2, 2).... ..SP$ (2, 5)
SP$ (1, 0) ; SP$ (1, 1) ; SP$ (1, 2).... ..SP$ (1, 5)
SP$ (0, 0) ; SP$ (0, 1) ; SP$ (0, 2).... ..SP$ (0, 5)

Mit einem solchen, zweidimensionalen Feld können wir jeden Studenten in der Klasse lokalisieren. Aber wie könnte man einen Studenten lokalisieren, der auf dem gleichen Platz wie Fritz Walter, aber in der nächsten Klasse, sitzt? Natürlich müssen wir dann neben Reihe und Spalte der Sitze noch die Nummer der Klasse angeben. Damit bekommt unser Feld noch eine weitere Dimension. Es gibt nämlich 12 Klassenräume im Schulgebäude. Es stehen verschiedene Wege zur Verfügung, um dieses Problem zu lösen. Der zweite ist, die Räume nach Stockwerken anzuordnen. 1. Raum im 1. Stock, 1. Raum im 2. Stock usw. Hier brauchen wir dann zwei weitere Dimensionen (insgesamt: Reihe, Spalte, Raum, Stock).

Angenommen, Fritz's Klassenraum wäre der 3. Raum im 2. Stock. Nach der ersten Methode könnten wir Fritz's Platz dann mit SP$ (N, R, C) angeben. Dabei ist: N die Zahl der Räume, R die Reihe und C die Spalte. Jupp sitzt dann z. B. in SP$ (7, 1, 3) d. h. in Raum 7, Reihe 1, Spalte 3.

Benutzen wir die zweite Methode, bekommen wir: SP$ (F, N, R, C). Dabei ist: F die Nummer des Stockwerks, N die Nummer des Raumes in diesem Stock, R die Reihe und C die Spalte. Um Fritz nun anzusprechen, müßten wir sagen: SP$ (2, 3, 1, 3) d. h.: 2. Stock, 3. Raum, 1. Reihe, 3. Spalte.

Die Anzahl der Dimensionen steigt an, wenn wir noch mehr Studenten in unsere Systematik aufnehmen wollen. Wir können noch weitere Dimensionen einführen für die Gebäude, die Universitäten, Städte, Länder usw.

In jedem Video-Genie-System wird die Anzahl der möglichen Dimensionen nur durch den vorhandenen Speicherplatz begrenzt.

26. Behandlung von Zeichenketten

Zeichenketten haben eine große Bedeutung in der Datenverarbeitung. Ein Computer, der keine Zeichenketten verarbeiten kann, ist nichts weiter als ein sehr leistungsfähiger Taschenrechner. Dieser Erkenntnis folgend, stehen Ihnen im Video-Genie-System außer den arithmetischen Funktionen, viele leistungsfähige Zeichenkettenmanipulations-Funktionen zur Verfügung.

In diesem Kapitel werden wir diese Zeichenkettenmanipulations-Statements, die in unserer erweiterten BASIC-Programmiersprache benutzbar sind, erläutern. Wir werden von nun an Zeichenketten mit ihrem englischen Namen bezeichnen, nämlich als ★★Strings★★.

Vergleichen von Strings (Zeichenketten)

Mit den relationalen Operatoren (=, <, >, usw.) können Strings auf ihre Gleichheit überprüft und nach ihrer alphabetischen Größe verglichen werden. Wird auf Gleichheit geprüft, so müssen alle Zeichen, führende und abschließende Leerzeichen (blanks) eingeschlossen, übereinstimmen, sonst wird auf ungleich entschieden.

Beispiel:

```
100 IF A$ = "JA" THEN 250
```

Strings werden Zeichen für Zeichen, von links nach rechts, verglichen. Um genau zu sein: die ASCII-Darstellung der Zeichen wird verglichen. Ein Zeichen mit einer höheren Codenummer ist größer als eines mit einer niedrigeren. Mit anderen Worten „AC" größer „AB". Für Buchstaben gilt die alphabetische Reihenfolge (A = Min, Z = Max). Werden Strings verschiedener Länge verglichen, so wird der kürzere String als kleiner angenommen, auch wenn seine Zeichen identisch mit denen des längeren sind. Daher ist „B" kleiner „B ". Die folgenden relationalen Operationen können zum Vergleichen von Strings eingesetzt werden:

>, <, <=, =>, =, <>

String Operation

Es gibt nur eine String Operation. Das ist das Aneinanderhängen von Strings. Der Operator ist das „+"-Zeichen.

Beispiel:

```
10 S1$ = "DIE SONNE"
20 S2$ = "SCHEINT"
30 S3$ = ","
40 C$ = S1$ + S2$ + S3$ + S2$ + S3$ + S2$ + "."
50 PRINT C$
60 END

READY
>RUN

DIE SONNE SCHEINT, SCHEINT, SCHEINT.
```

ASC (String)

Dieses Statement berechnet den ASCII-Code des ersten Zeichens des angegebenen Strings. Der String muß in Anführungszeichen eingeschlossen sein. Ein leerer String als Argument erzeugt eine Fehlermeldung.

Beispiel:

```
100 PRINT "DER ASCII-CODE VON 'H' IST : " ;, ASC ( "H" )
105 S$ = "HEIM" : PRINT "DER STRING HEISST :" ; S$
110 PRINT "DER ASCII-CODE DES 1. BUCHSTABENS IST : " ; ASC ( S$)
120 END

READY
>RUN

DER ASCII-CODE VON 'H' IST : 72
DER STRING HEISST : HEIM
DER ASCII CODE DES 1. BUCHSTABENS IST : 72.
```

Beide Zeilen drucken die gleiche Zahl.

Eine Liste der ASCII-Codes finden Sie in Anhang F.

CHR$ (Ausdruck)

Dieses Statement macht das genaue Gegenteil der ASC Funktion. Es erzeugt das zum Wert von Ausdruck gehörige ASCII-Zeichen. Als Argument darf eine Zahl von 0 bis 255, oder ein Ausdruck mit diesem Wert, auftreten. Das Argument muß in Klammern gesetzt werden.

```
100 PRINT CHR$( 33 ): REM SCHREIBE EIN AUSRUFEZEICHEN
```

LEFT$ (String, n)

Das Statement erzeugt die ersten n Zeichen des angegebene Strings. String und n müssen in Klammern stehen. String kann eine Zeichenkettenvariable oder Konstante sein, n kann ein numerischer Ausdruck oder Konstante sein.

Beispiel:

```
10 A$ = "ABCDEFG"
20 B$ = LEFT$ ( A$, 4 )
30 PRINT B$
40 END

READY
>RUN

ABCD
```

RIGHT$ (String, n)

Erzeugt die n letzten Zeichen von String. Die Argumente müssen in Klammern stehen. String kann eine Zeichenkettenvariable oder Konstante sein, n kann eine numerische Variable oder Konstante sein. Ist die Länge von String kleiner oder gleich n, wird der gesamte String erzeugt.

Beispiel:

```
10 A$ = "ABCDEFG"
20 B$ = RIGHT$ ( A$, 3 )
30 PRINT B$
40 END

READY
>RUN

EFG
```

LEN (String)

Gibt die Länge von String an. String kann Variable, Konstante oder Zeichenkettenausdruck sein.

Beispiel:

```
10 A$ = "ABCDEFG"
20 PRINT "DIE LÄNGE DER ZEICHENKETTE : " ; LEN ( A$ )
30 END

READY
>RUN

DIE LÄNGE DER ZEICHENKETTE : 7
```

MID$ (String, p, n)

Erzeugt einen Teilstring von String. Er beginnt an Position p und ist n Zeichen lang. Die Argumente müssen in Klammern stehen. String kann Konstante, Variable oder Ausdruck sein, p und n können numerische Konstante, Variable oder Ausdruck sein.

Beispiel:

```
10 A$ = "ABCDEFG"
20 B$ = MID$ ( A$, 3, 4 )
30 PRINT "DER NEUE STRING HEISST : " ; B$
40 END

READY
>RUN

DER NEUE STRING HEISST : CDEF
```

STR$ (Ausdruck)

Verwandelt eine numerische Konstante oder einen Ausdruck in einen String. Das Angebot muß in Klammern stehen.

Beispiel:

```
10 A = 34.56
20 B$ = STR$ ( A )
30 B$ = B$ + "%"
40 PRINT "DAS ERGEBNIS IST : " ; B$

READY
>RUN

DAS ERGEBNIS IST : 34.56%
```

STRING$ (n, Zeichen oder Zahl)

Erzeugt einen String, der aus n mal dem Zeichen besteht.

Beispiel:

```
10 PRINT STRING$ ( 10, "★" )
20 END

READY
>RUN
★★★★★★★★★★
```

An Stelle von Zeichen kann man auch eine Zahl von 0 bis 255 angeben. Sie wird als ASCII-Code behandelt und das entsprechende Zeichen oder der Graphische Code werden erzeugt.

```
10 PRINT  STRING$ ( 10, 35 )
20 END

READY
>RUN
# # # # # # # # # #
```

VAL (String)

Macht das Gegenteil der STR$ Funktion. Erzeugt einen numerischen Wert aus String.

Beispiel:

```
10 A$ = "56"
20 B$ = "23"
30 C = VAL ( A$ + "." + B$ )
40 PRINT "DAS ERGEBNIS IST : " ; C ; "," ; C + 100
50 END

READY
>RUN

DAS ERGEBNIS IST : 56.23, 156.23
```

27. ARITHMETISCHE FUNKTIONEN

In diesem Kapitel werden wir die fest programmierten, arithmetischen Funktionen des Video-Genie-Systems besprechen. In den meisten Fällen muß ein Argument an die Funktion weitergegeben werden, bevor der Funktionswert berechnet werden kann. Dieses Argument kann eine numerische Konstante, eine numerische Variable oder ein numerischer Ausdruck sein. Das allgemeine Format ist:

Ergebnis = Funktion (Argument)

Beispiel:

```
10 A = RND ( 3 )
20 B = INT ( C )
30 E = SQR ( F★G—4 )
```

Wir behandeln folgende Funktionen:

1. ABS (X)
2. ATN (X)
3. CDBL (X)
4. CINT (X)
5. COS (X)
6. CSNG (X)
7. EXP (X)
8. FIX (X)
9. INT (X)
10. LOG (X)
11. RANDOM
12. RND (X)
13. SGN (X)
14. SIN (X)
15. SQR (X)
16. TAN (X)

ABS (X)

Berechnet den Absolutwert von X.

ATN (X)

Berechnet den Arcustangnes von X (im Bogenmaß). Um das Ergebnis in Grad zu erhalten, multipliziere ATN (X) mit 57.29578.

CDBL (X)

Erzeugt die doppelt genaue Darstellung von X. Das Ergebnis erthält 17 Stellen, wovon die Stellen, die das Argument enthalten signifikant sind.

CINT (X)

Berechnet die nächste, ganze Zahl, die kleiner als das Argument ist. Das Argument muß zwischen —32768 und +32767 liegen.
Beispiel: CINT (2.6) = 2; CINT (—2.6) = 3

COS (X)

Berechnet den Cosinus des Arguments (im Bogenmaß). Soll er in Grad berechnet werden: COS (X★.0174533)

CSNG (X)

Erzeugt die einfache genaue Darstellung von X. Berechnet eine 6stellige Zahl mit 4/5 Rundung bei doppelt genauem X.

EXP (X)

Berechnet die Exponential-Funktion von X, d. h. e hoch X. Das ist die Umkehrfunktion zu LOG (X).

FIX (X)

Trennt die Nachkommastellen von X ab. FIX (1.5) = 1; FIX (—1.5) = —1

INT (X)

Erzeugt die ganzzahlige Darstellung von X. Berechnet die größte, ganze Zahl, die nicht größer als X ist.

Beispiel: INT (3.5) = 3 ; INT (—3.5) ist gleich —4

LOG (X)

Berechnet den natürlichen Logarithmus von X. d. h. log_e(X). Um einen Logarithmus zu einer anderen Basis zu berechnen benutzen Sie folgende Formel:
log_b(X) = log $(X)_e$/log_e(b) (b ist die Basis)

RANDOM

Führt der Computer diese Funktion aus, so erzeugt er einen neuen Vorrat an Zufallszahlen. Diese Funktion benötigt keine Argumente.

RND (X)

Erzeugt eine Pseudozufallszahl aus der aktuellen Menge von Zufallszahlen. (Werden intern generiert, kein Zugriff vom Benutzer möglich).
RND (0) erzeugt eine einfach genaue Zufallszahl zwischen 0 und 1
RND (X) erzeugt eine ganze Zahl zwischen 1 und X
X muß kleiner als 32768 und positiv sein

SGN (X)

Die Vorzeichenfunktion. Sie erzeugt eine —1, wenn X negativ ist, Null, wenn X Null ist und +1, wenn X positiv ist.

SIN (X)

Berechnet die Sinusfunktion von X (im Bogenmaß)
Der Sinus in Grad: SIN (X★.0174533)

SQR (X)

Berechnet die Quadratwurzel von X.

TAN (X)

Berechnet die Tangensfunktion von X. (im Bogenmaß)
TAN (X) in Grad: TAN (X★.0174533).

28. Zusätzliche Basic-Befehle

INP (Portnummer)

Liest einen 8-bit Wert aus dem angegebenen Port. Das Video-Genie-System kann 256 Ports adressieren. Sie werden von 0 bis 255 numeriert. Einige Ports sind schon intern verwendet.

Beispiel:

```
10 A = INP ( 124 )
```

Liest 8-bit Wert aus Port Nr. 124 nach A.

OUT Portnummer, Wert

Gibt einen 8-bit Wert auf dem angegebenen Port aus. Dieses Statement verlangt zwei Argumente: die Portnummer und den Wert, der an diesem Port ausgegeben werden soll. Das Video-Genie-System kann 256 Ports adressieren, sie sind von 0 bis 255 durchnumeriert.

Beispiel:

```
30 OUT 14,240
```

Gibt den Wert 240 an Port 14 aus. Beide Argumente müssen zwischen 0 und 255 liegen.

PEEK (Adresse)

Diese Funktion holt einen 8-bit Wert aus der Speicherzelle, die mit Adresse dezimal angegeben wurde. Der Wert von PEEK ist ebenso dezimal und liegt zwischen 0 und 255.

Beispiel:

```
20 B = PEEK ( 3000 )
```

Schreibt den Inhalt von Speicherzelle 3000 nach B.

POKE Adresse, Wert

Dieses Statement schreibt einen 8-bit Wert in die, durch Adresse dezimal angegebene Speicherzelle. Es braucht zwei Argumente: Adresse und Wert. Wert muß zwischen 0 und 255 liegen.

Beispiel:

```
10 A = 250
20 POKE 19000, A : REM SCHREIBE A IN ZELLE 19000
30 B = PEEK ( 19000 ) : REM HOLE DEN WERT VON ZELLE 19000
40 PRINT "DAS ERGEBNIS IST : " ; B
50 END
```

MEM

Gibt die Zahl der nicht benutzten Bytes im Speicher an.

Beispiel:

```
200 IF MEM < 180 THEN 700
```

Wenn MEM als Kommando eingesetzt werden soll, muß es zusammen mit PRINT ausgeführt werden. PRINT MEM gibt die Zahl der Bytes im Speicher aus, in denen kein Programm, Variablen, Zeichenketten usw. stehen.

INKEY$

Liest ein Zeichen von der Tastatur. Wurde zum Zeitpunkt der Ausführung der INKEY Funktion keine Taste betätigt, so wird ein Leerstring erzeugt (=" "). Die von der INKEY$ Funktion eingelesenen Daten erscheinen nicht auf dem Schirm.

Beispiel:

```
10 REM EINGABE EINES PASSWORDS
20 REM OHNE ES AUF DEM SCHIRM SICHTBAR ZU MACHEN
30 CLS
40 PRINT "GIB PASSWORD EIN"
50 A$ = INKEY$ : IF A$ = "O" THEN 60 ELSE 50
60 B$ = INKEY$ : IF B$ = "K" THEN 70 ELSE 60
70 PRINT "WILLKOMMEN"
```

Beispiel:

```
10 A$ = INKEY$ : IF A$ = " " THEN 10
```

Dieses Programm wartet, bis eine Taste betätigt wurde.

POS (dummy argument)

Es wird eine Zahl von 0 bis 39, die die momentane Position des Cursors angibt, erzeugt. „dummy argument" kann irgendeine Zahl sein, meist wird die „0" benutzt.

Beispiel:

```
100 A = POS ( 0 )
```

USR (argument)

Ruft ein Unterprogramm in Maschinensprache auf. Das Unterprogramm kann mit POKE erzeugt oder vom Band eingelesen werden. Wenn der Benutzer in der Programmierung in Maschinensprache noch nicht sicher ist, ist vom Einsatz dieser Funktion abzuraten.

Die Startadresse des Unterprogramms wird mit POKE in die Adressen 16526 und 16527 mit dem am wenigsten signifikanten Byte in 16526, geschrieben.

Um ein Argument in das Unterprogramm zu übergeben, sollte das Unterprogramm an seinem Anfang ein CALL 0A7F'H (2687 in dezimal), ausführen. Das Argument wird dann in das HL-Register geschrieben.

Um ins BASIC zurück zu kommen, ohne einen Wert zurückzugeben, wird das Unterprogramm mit RET abgeschlossen. Soll ein Wert zurückgegeben werden, so wird er in das HL-Registerpaar geladen und am Ende des Unterprogramms ein JP 0A9A'H (=2714 in dezimal) ausgeführt. Die Werte werden als 2 Byte, ganze Vorzeichenzahlen behandelt. Die USR Funktion reserviert 8 Stack Ebenen für das Unterprogramm.

Das Unterprogramm sollte in den obersten Speicherbereich geschrieben werden. Um es davor zu schützen, daß es vom Basic als Programmspeicher betrachtet und damit sein Inhalt zerstört wird, sollte der Benutzer, wenn das System nach dem Einschalten MEM SIZE? ausgibt, die höchste Speicheradresse, die das BASIC noch benutzen darf, ohne sein Unterprogramm zu überschreiben, angeben (in dezimal).

Beispiel:

```
10 INPUT I% : REM ARGUMENT EINGEBEN
15 REM ★ STARTADRESSE EINSCHREIBEN
20 POKE 16526, 0 : POKE 16527, 120
30 A = USR ( 1% ) : REM ★ A WIRD ZURÜCKGEGEBEN★
```

VARPTR (variablen name)

Die Adresse, an der der Wert der Variablen im Speicher steht, wird berechnet.

10 K = VARPTR (A)

Für die verschiedenen Variablentypen bedeutet K dann folgendes:

(i) 2-Byte, ganzzahlige Variablen (A%)

K = LSB
K + 1 = MSB

(ii) Variablen einfacher Genauigkeit (A)

K = LSB
K + 1 = Nächstes MSB
K + 2 = MSB
K + 3 = Exponent

(iii) Variablen doppelter Genauigkeit (A#)

K = LSB
K + 1 = nächstes MSB
.
.
.
K + 6 = MSB
K + 7 = Exponent

(iv) Zeichenketten Variablen (string) (A$)

K = Länge des Strings
K + 1 = LSB der Anfangsadresse des Strings
K + 2 = MSB der Anfangsadresse des Strings

LSB bezeichnet das am wenigsten signifikante Byte.
MSB bezeichnet das am höchsten signifikante Byte.

29. Hochauflösende Grafik

Neben dem Text-Modus, in dem Sie Programme eingeben etc., gibt es beim Colour-Genie den Modus für hochauflösende Grafik. In diesem Modus steht Ihnen eine Auflösung von 160 mal 96 Punkten zur Verfügung. Jeder der Punkte kann eine von 4 Farben annehmen.

Der Punkt mit den Koordinaten 0,0 liegt in der linken oberen Bildschirmecke. Die X-Koordinaten nehmen nach rechts hin zu und die Y-Koordinaten nach unten hin.

Sie können, wie im Kapitel über die Tastatur beschrieben, durch gleichzeitiges Drücken von <CTRL> und <MOD SEL> direkt den Inhalt des Bildschirmspeichers für hochauflösende Grafik sichtbar machen. Zurück kommt man mit <BREAK>.

Interessanter ist natürlich die Umschaltung vom Programm aus. Diese geschieht mit den 2 Befehlen „LGR“ und „FGR“.

Auch die weiteren Grafik-Befehle wollen wir nun kennenlernen. Anschließend finden Sie einige Programmbeispiele, anhand derer die Grafikbefehle demonstriert werden.

LGR

Umschalten auf den Textmodus.

FGR

Umschalten auf den Grafikmodus.

FCLS

Dieser Befehl löscht den Bildschirmspeicher für hochauflösende Grafik.

BGRD

Mit „BGRD“ wird im Grafik-Modus die Hintergrundfarbe (magenta) eingeschaltet.

NBGRD

Dieser Befehl schaltet die Hintergrundfarbe wieder ab. Dabei wird ein eventueller Bildinhalt nicht gelöscht.

Fill n

Dieser Befehl malt den gesamten Bildschirminhalt mit der durch n angegebenen Farbe aus. Der Bildschirminhalt wird dabei gelöscht. Dabei kann n den Wert 1, 2, 3 oder 4 annehmen, der folgender Farbe entspricht:
n = 1: schwarz bzw. magenta wenn „BGRD“
n = 2: blau
n = 3: orange
n = 4: grün
Dieser Befehl löscht den Inhalt des Grafikspeichers.

FCOLOUR n

Mit „FCOLOUR“ legen Sie die Farbe fest, in der alle weiteren Punkte gezeichnet werden. Dabei entspricht der Wert von n folgenden Farben:
n = 1: schwarz oder, wenn Hintergrundfarbe angeschaltet, magenta.
n = 2: blau
n = 3: orange
n = 4: grün

CIRCLE X, Y, R

Dieser Befehl zeichnet einen Kreis, dessen Mittelpunkt die Koordinaten X und Y hat und dessen Radius R beträgt. Überschreitet einer der 3 Werte 255 wird eine Fehlermeldung ausgegeben. Kreise, die über den Bildschirm hinausgehen, stimmen nur teilweise in ihrer Darstellung.

PLOT

Dieser Befehl hat mehrere Formate:
1. PLOT X,Y setzt den Punkt mit den Koordinaten X und Y.
2. PLOT X,Y TO X1, Y1 zieht eine Linie von Punkt X,Y zum Punkt X1,Y1
3. PLOT X,Y TO X1,Y1 TO X2,Y2 TO . . .
   zieht eine Linie von Punkt X,Y zum Punkt X1,Y1 und von dort zum Punkt X2,Y2 usw.

NPLOT

Dasselbe wie „PLOT“, nur daß die Linie zurückgesetzt wird.

N = CPOINT(X,Y)

Hiermit können Sie die Farbe des Punktes mit den Koordinaten X und Y feststellen. N nimmt dabei folgende Werte an:
N = 0 für schwarz bzw. Hintergrundfarbe,
N = 1 für blau,
N = 2 für orange,
N = 3 für blau
Beachten Sie, daß bei diesem Befehl die Koordinaten in Klammern angegeben werden.

PAINT X,Y,C,G

Dieser Befehl dient dazu, Flächen auszumalen. Ausgemalt wird von dem Punkt mit den Koordinaten X und Y aus. Dabei wird die Farbe C benutzt und dann aufgehört, wenn die Farbe G erreicht wird. Die Werte von C und G müssen denen, die Sie vom „FCOLOUR“-Befehl her kennen, entsprechen.

Hierzu gleich das 1. Programmbeispiel: Es soll einen blauen Kreis auf schwarzem Hintergrund zeichnen, mit dem Mittelpunkt bei Koordinate X = 70 und Y = 30, und dem Radius R = 30. Dieser Kreis soll dann orange ausgemalt werden.

Die „REM“-Befehle schließen eine Erklärung an die jeweiligen Befehle an, Sie müssen diese natürlich nicht mit eingeben.

```
>10 FCLS : REM  Löschen des Grafik-Speichers.
>20 FGR : REM  Umschalten in den Grafik-Modus.
>30 FCOLOUR 2 : REM  Farbe Blau für den Kreis einstellen.
>40 CIRCLE 70,30,30 : REM  Kreis zeichnen.
>50 PAINT 70,30,3,2 : REM  Kreis ausmalen.
>60 GOTO60 : REM  Endlosschleife, damit die Grafik sichtbar bleibt.
```

2. Beispiel: Sie wollen ein Dreieck mit den Eckpunkten P(80,10); Q(150,90) und S(10,90) in Blau auf orangem Hintergrund zeichnen:

```
>10 FCLS : REM  Löschen des Grafik-Speichers.
>20 FGR : REM  Umschalten in den Grafik-Modus.
>30 FILL 3 : REM  Bildschirm in Orange ausmalen.
>40 FCOLOUR 2 : REM  Farbe Blau einstellen.
>50 PLOT 80, 10 TO 150,90 TO 10,90 TO 80,10 : REM  Dreieck
>60 GOTO 60 : REM  Endlosschleife.
```

3. Beispiel: Zeichnen einer Sinuskurve (orange) mit X- und Y-Achse (blau):

```
>10 FCLS : REM  Grafikspeicher löschen.
>20 FGR : REM  Umschalten in den Grafik-Modus.
>30 FCOLOUR 2 : REM , Farbe Blau für Achsen einstellen.
>40 PLOT 0,0 TO 0, 95 : REM  Y-Achse zeichnen.
>50 PLOT 0,48 TO 159,48  : REM  X-Achse zeichnen.
>60 FCOLOUR 3 : REM  Farbe Orange für Sinuskurve einstellen.
>70 FOR X = 0 TO 159 : REM  Schleife für Sinus-Werte.
>80 Y=SIN ( X / 20 ) ★47+48 : REM  Y-Werte aus Sinus-Funktion.
>90 PLOT X,Y : REM  Punkt setzen.
>100 NEXT X : REM  Nächster X-Wert (bis 159).
>110 GOTO 110 : REM  Endlosschleife.
```

Wollen Sie die Kurve ausgemalt haben, so geben Sie

```
>90 PLOT X,48 TO X,Y
```

ein.

4. Beispiel: Zeichnen von zufälligen Bildern mit dem „PLOT"-Befehl (Kommentare sind hier weggelassen, da keine neuen Befehle vorkommen.):

```
>10 FCLS : FGR
>20 X=RND ( 159 ) : Y=RND ( 95 ) : S=RND ( 8 )+1
>30 FCOLOUR RND ( 4 )
>40 FOR A=0 TO 159 STEP S : PLOT X,Y TO A,0 : NEXT A
>50 FOR A=0 TO 95  STEP S : PLOT X,Y TO159 ,A : NEXT A
>60 FOR A=159 TO 0 STEP —S : PLOT X,Y TO A,95 : NEXT A
>70 FOR A=95 TO 0 STEP —S : PLOT X,Y TO 0,A : NEXT A
>80 GOTO 20
```

5. Beipiel: Lissajou'sche Figuren:

```
>10 CLS
>20 INPUT"Frequenzverhältnis ( 0,25 bis 3 ) " ; FV
>30 FCLS : FGR : FCOLOUR4
>40 X=80 : X=48
>50 W=W+0.1
>60 X1=X : Y1=Y
>70 X=80+SIN ( W )★78
>80 Y=47+SIN ( W★FV ) ★47
>90 PLOT X,Y TO X1,Y1
>100 GOTO 50
```

30 Die Befehle „SHAPE“ und „SCALE“

Genaugenommen handelt es sich bei diesen Befehlen auch um Befehle für die hochauflösende Grafik. Da die Handhabung des „SHAPE“-Befehls aber recht kompliziert ist wird diesem zusammen mit dem dazugehörigen „SCALE“-Befehl dieses separate Kapitel gewidmet.

SHAPE X,Y

Der „SHAPE“-Befehl ermöglicht es, komplexe Figuren in hochauflösender Grafik an beliebiger Bildschirmstelle mit einem einzigen Kommando darzustellen.

Die Gestalt und Farbe der Figur werden dabei vorab in der sog. „Shape Table“ definiert. Darunter hat man sich einen reservierten Speicherbereich vorzustellen, in dessen einzelnen Speicherplätzen jeweils ein Teil der Figur festgelegt ist. Die Festsetzung der einzelnen Werte geschieht bitweise für Richtung und Farbe.

Sollten Sie mit den Begriffen „Bit“, „Byte“ und „hexadezimal“ noch nicht vertraut sein, lesen Sie am besten vorab das folgende Kapitel, das diese Begriffe erklärt.

Bei einem Rechner mit 16K RAM beginnt die Shape Table bei Adresse 7F00H, bei einem Rechner, dessen RAM auf 32K aufgerüstet ist, beginnt die Shape Table bei Adresse BF00H. Wenn Sie beim Einschalten als Antwort auf die „MEM SIZE?“-Frage durch Eingabe einer Adresse in dezimal Speicher reserviert haben, werden die ersten 256 Byte im reservierten Bereich dennoch der Shape Table zugewiesen. In den beiden Speicherzellen 40B1H und 40B2H steht der Wert, der angibt, wo die Shape Table beginnt, minus 1.

Wie wird eine Figur programmiert?

Dazu sollten Sie folgendermaßen vorgehen:

Zuerst zeichnen Sie am besten auf einem Stück Papier die Umrisse der Figur auf, die Sie definieren wollen. Dabei müssen Sie beachten, daß eine Figur, die vom „SHAPE“-Befehl gezeichnet werden soll, nur aus geraden Linien bestehen darf. Ferner muß die Figur in einem Zug gezeichnet werden, so als ob Sie mit einem Stift eine Zeichnung machen, ohne dabei abzusetzen.

Wie schon gesagt, besteht eine Shape-Figur nur aus geraden Linien. Jede Linie hat dabei eine feste Länge, man muß also seine Figur in feste Längeneinheiten unterteilen.

Auch die Farbe der einzelnen Linien wird mit festgelegt, daher sollten sie sich auch die Farbe überlegen.

Außerdem müssen Sie die Anzahl der einzelnen Linien zählen, da diese die Länge der Shape Table ist.

Jedes Byte legt zwei Linien gleichzeitig fest, dabei gilt Format:

	1. Linie				2. Linie			
Bit Nr.	B7	B6	B5	B4	B3	B2	B1	B0
Wert des Bits	128	64	32	16	8	4	2	1

B5 und B4 geben die Richtung für die erste Linie an, B1 und B0 die Richtung für die zweite Linie.
Für die 4 Richtungen nehmen B5 und B4 bzw. B1 und B0 folgende Werte an:

Nach . . .	B1 (B5)	B0 (B4)
Rechts	0	0
Unten	0	1
Links	1	0
Oben	1	1

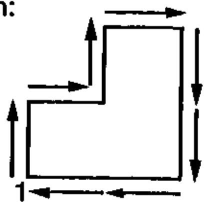
B7 und B6 geben die Farbe der ersten Linie an, B3 und B2 die Farbe für die zweite Linie.
Dabei gilt:

Farbe	B7 (B3)	B6 (B2)	
Schwarz	0	0	<— Magenta, wenn „BGRD“
Blau	0	1	
Orange	1	0	
Grün	1	1	

In Speicherzelle 7F00H (bzw. BF00H bei 32K RAM) wird die Länge (gleich Anzahl der Linien durch zwei) geschrieben. In die folgenden Speicherzellen werden dann die Bytes gemäß obigem Format für die einzelnen Linien geschrieben.

Dies alles will ich nun anhand eines Beispiels erklären.

Angenommen Sie wollen folgende Figur definieren:



Alle Linien sollen dabei grün sein.

Sie setzen also bei Punkt eins an und dann:
Nach oben, rechts, oben, rechts, unten, unten, links, links
Die Bytes sehen also so aus (Adressen in Klammer: bei 32K RAM)

Adresse	Wert in binär								und dezimal (Länge)
7F00H (BF00H)	0	0	0	0	0	1	0	0	4
7F01H (BF01H)	1	1	1	1	1	1	0	0	252
7F02H (BF02H)	1	1	1	1	1	1	0	0	252
7F03H (BF03H)	1	1	0	1	1	1	0	1	221
7F04H (BF04H)	1	1	1	0	1	1	1	0	238

Folgendes Programm, das die Figur definiert, sieht also so aus (Programm für 16K, bei 32K RAM entsprechende Werte einsetzen):

```
>10 POKE &H7F00,4
>20 POKE &H7F01,252
>30 POKE &H7F02,252
>40 POKE &H7F03,221
>50 POKE &H7F04,238
>60 FGR : FCLS
>70 SHAPE 80,48
>80 GOTO 80
```

Programmerklärung:

Zeile 10...50 definiert die Figur.
Zeile 60 löscht den Bildschirm und schaltet in den Grafik-Modus.
Zeile 70 zeichnet die definierte Figur, wobei bei Bildschirmkoordinate 80,48 begonnen wird. Hier können Sie natürlich auch andere Koordinaten einsetzen.
Zeile 80 bewirkt, das die Grafik erhalten bleibt.

Wenn Sie das Programm starten, sehen Sie, daß die Figur sehr klein gezeichnet wird.

Da schafft der „SCALE"-Befehl Abhilfe.

SCALE n

Mit diesem Kommando legen Sie den Vergrößerungsfaktor für die Zeichnung mit dem „SHAPE"-Befehl fest.

Fügen Sie z. B. in obiges Programm diese Zeile ein:

```
>65 SCALE 10
```

Die Figur erscheint in zehnfacher Größe.

Der maximale Wert von n ist 255.

NSHAPE X,Y

Dasselbe wie SHAPE X,Y, nur daß die Figur zurückgesetzt wird.

XSHAPE X,Y

Dasselbe wie Shape X, Y, nur, daß die Bits für die Farbe komplementiert werden, d. h. die Farben ändern sich folgendermaßen:

Schwarz → Grün
Blau → Orange
Orange → Blau
Grün → Schwarz

31. Bit, Byte, Hexadezimalzahl

Diesen 3 Begriffen werden Sie im Folgenden noch öfter begegnen. Daher seien sie hier kurz erklärt.

Das „Herz" Ihres Computers ist ein Z80-Mikroprozessor. Ein Mikroprozessor selbst versteht kein Basic sondern die sogenannte Maschinensprache. Der eingebaute Basic-Interpreter übersetzt ein Basic-Programm in die für den Mikroprozessor verständliche Maschinensprache.

Die Maschinensprache arbeitet auf binärer Ebene. Die grundlegende Informationseinheit dabei ist ein „Bit". Ein Bit kann nur 2 Zustände annehmen: 0 und 1, an und aus, Transistor leitet oder nicht . . .

Nun faßt man 8 dieser Bits zu einem „Byte" zusammen. Die Z80 CPU ist ein 8-Bit-Mikroprozessor, d. h. sie kann jeweils 8 Bits = 1 Byte auf einmal verarbeiten.

Da ein Byte 8 Bit enthält, sind 2^8 = 256 Zustände möglich (von 0...255 einschließlich).

Da die Darstellung eines Bytes in Bit-Schreibweise (z. B. 01101111 binär = 111 dezimal) recht umständlich ist, nimmt man je 4 Bit aus einem Byte und gibt diese dann durch ein Hexadezimaläquivalent aus:

Binär:	Hex.:
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Ein Byte, das, wie schon gesagt, aus 8 Bit besteht, wird dann durch 2 Hexadezimalziffern dargestellt.

Z. B.: 0111 1110 = 7E Hex. = 126 dez.

Der Z80-Mikroprozessor kann maximal 2^{16} = 65536 Speicherzellen adressieren. Jede Speicherzelle entspricht einem Byte, das 256 verschiedene Zustände annehmen kann.

Die Adresse einer Speicherzelle wird durch 16 Bits angegeben. Hier werden 4 mal 4 Bits zu je einer Hexadezimalziffer zusammengefaßt.

Ein Beispiel:

0110 1111 1100 1110 = 6FCE Hex. = 28622 Dez.

Die Adressen reichen also von 0000H bis FFFFH.

Wie im nächsten Kapitel beschrieben, kann das Genie-Basic Hexadezimalzahlen direkt in Dezimalzahlen umrechnen. Umgekehrt ist dies nicht möglich, daher ist hier ein kleines Basic-Programm aufgelistet, das diese Aufgabe übernimmt:

```
>10 CLS
>20 DIM A$ ( 15 )
>30 FOR A = 0 TO 15
>40 READ A$ ( A )
>50 NEXT A
>60 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
>70 INPUT "Dezimalzahl" ; Z
>80 A$ = ""
>90 A=Z
>100 A1=FIX ( A / 16 )
>110 A2=A—A1★16
>120 A$=A$ ( A2 )+A$
>130 A=A1
>140 IF A>0 THEN 100
>150 IF LEN ( A$ ) < 4 THEN A$ = "0"+A$ : GOTO 150
>160 PRINT Z ; " dezimal entspricht " ; A$ ; " hexadezimal."
>170 GOTO 70
```

32. Spezial-Befehle

&HXXXX

„XXXX“ steht für eine 4stellige hexadezimale Zahl. Mit diesem Befehl kann man direkt hexadezimale Zahlen verarbeiten bzw. sie in dezimale Zahlen umrechnen. Z. B.:

```
>PRINT &H4F00
 20224
READY
>■
```

4F00 hexadezimal entspricht 20224 dezimal.

Geben Sie folgendes ein:

```
>PRINT &HF000
—4096
READY
>■
```

Wieso wurde dieses mal eine negative Zahl ausgegeben? Dies hat folgenden Grund: Der Computer gibt ab Hexadezimalzahl 8000H als Dezimaläquivalent eine negative Zahl aus. Der Vorteil dieser Methode liegt darin, daß alle Adressen mit Integer-Variablen adressiert werden können. Das Gleiche muß man selber bei den Befehlen „PEEK“ und „POKE“ beachten — auch hier muß man über 32767 (dezimal) einen negativen Wert eingeben, der sich wie folgt errechnen läßt:

```
>...IF X>32767 THEN X=X—65536
```

Die nachfolgende Speicherzelle auf 32767 (7FFFH) ist also mit —32768 (8000H) anzugeben — Die höchste mögliche Adresse ist —1 (FFFFH).

&OXXX

Rechnet die 3stellige Oktalzahl XXX in dezimal um. Z. B.:

```
>PRINT &O234
 156
READY
>■
```

CALL XXXX

Springt zur Adresse XXXX (4stellige Hexadezimalzahl). Dieser Befehl wird zum Aufruf von Maschinenspracheunterprogrammen benutzt.
Z. B.:

```
>CALL 01C9
```

springt die Bildschirm-Lösch-Routine in ROM an.

RENUM N,S

Dieser Befehl dient zur Umnummerierung von Basicprogrammzeilen. Nach der Umnummerierung beginnt das Programm bei Zeilennummer N und geht dann in Schritten von S weiter.
Ein Beispiel:
Sie haben folgendes Programm im Speicher:

```
1 REM Zeile 1
2 REM Zeile 2
3 REM Zeile 3
4 REM Zeile 4
```

Nun geben Sie ein:

```
>RENUM 2,20
>LIST
2 REM Zeile 1
22 REM Zeile 2
42 REM Zeile 3
62 REM Zeile 4
READY
>■
```

Wird nur „RENUM“ eingegeben, setzt das Basic automatisch für N und S den Wert 10 ein.

33. Der „CHAR“-Befehl

Das Colour-Genie bietet die Möglichkeit zwischen 4 verschiedenen Zeichensätzen zu wählen. Dies geschieht mit dem „CHAR n“-Befehl, wobei n die Werte 1, 2, 3 und 4 annehmen kann.

Folgende Tabelle gibt an, wie sich der Zeichensatz mit dem Wert von n ändert:

ASCII		Wert von n			
Code		1	2	3	4
32—127	(20H—7FH)	AZ	AZ	AZ	AZ
128—191	(80H—BFH)	PR1	PR1	GR1	GR1
191—255	(COH—FFH)	PR2	GR2	PR2	GR2

Erklärung der Abkürzungen:

AZ: Alphanumerische Zeichen, also Buchstaben, Zahlen und Sonderzeichen.

GR1: Erste Gruppe der fest programmierten Grafikzeichen. Diese Grafikzeichen sind nicht über die Tastatur zu erreichen, sondern nur mit PRINT CHR$ (X) oder POKE Y, X, mit X=128...191 und Y=&H4400...&H47FF.

GR2: Zweite Gruppe der fest programmierten Grafikzeichen. Diese Grafikzeichen können nach Drücken von <MOD SEL> gemäß Aufdruck auf der Vorderseite der Tasten erreicht werden.

PR1: Erste Gruppe der frei programmierbaren Zeichen.

PR2: Zweite Gruppe der frei programmierbaren Zeichen.

Folgendes Programm zeigt Ihnen die 4 verschiedenen Zeichensätze nach entsprechender Zahleneingabe:

```
>10 CLS
>20 INPUT “ZEICHENSATZ ( 1...4 )“ ; N
>30 IF N<1 OR N>4THEN GOTO 20
>40 COLOUR 9
>50 CLS
>60 CHAR N
>70 FOR A = 0 TO 255
>80 POKE &H4400 + A★2, A
>90 NEXT
>100 COLOUR 5
>110 PRINT @ 880,“<RETURN>für Neustart“ ;
>120 INPUT X$
>130 RUN
READY
>■
```

34. Programmierbare Zeichen

Wie Sie im vorigen Kapitel gesehen haben, gibt es bis zu 128 programmierbare Zeichen; die volle Menge von 128 definierbaren Zeichen steht nach Ausführung von „CHAR 1“ zur Verfügung.

Mit den programmierbaren Zeichen kann man eine Grafikauflösung von 320 mal 192 Punkten erzielen. Außerdem hat Grafik über programmierte Zeichen den Vorteil, daß man sie beliebig mit Text mischen kann.

Wenn der Computer neu eingeschaltet ist, wird bei Abruf eines der definierbaren Zeichen ein Leerzeichen ausgegeben.

Jedes definierbare Zeichen besteht aus einer 8 mal 8 Matrix, also einem Feld von 8 mal 8 Punkten. Der Speicher für die programmierbaren Zeichen liegt von F400H bis F7FFH. Dabei entspricht jedes gesetzte Bit in einem Byte einem Punkt in einer Zeile des Zeichens. Ein Zeichen besteht aus 8 Zeilen, daher sind insgesamt 8 Bytes für ein Zeichen notwendig.

Als Beispiel wollen wir das Zeichen mit dem ASCII-Code 128, erreichbar mit PRINT CHR$(128), betrachten. Dies ist das erste der programmierbaren Zeichen, also belegt es den Speicher von F400H bis F407H einschließlich. Eine Skizze soll dies verdeutlichen:

	Bit-nummer	7	6	5	4	3	2	1	0
Adressen (hex.)	F400	x	x	x	x	x	x	x	x
(für ASCII-Zeichen	F401	x	x	x	x	x	x	x	x
Nummer 128)	F402	x	x	x	x	x	x	x	x
	F403	x	x	x	x	x	x	x	x
	F404	x	x	x	x	x	x	x	x
	F405	x	x	x	x	x	x	x	x
	F406	x	x	x	x	x	x	x	x
	F407	x	x	x	x	x	x	x	x
2 hoch Bitnummer = Wert des Bits		128	64	32	16	8	4	2	1

Jedes „x“ stellt also ein Bit in dem Byte dar, dessen Adresse links angegeben ist. Jeder Punkt, der in einer Zeile gesetzt werden soll, entspricht also einem Bit. Jedes Bit, das in einer Zeile gesetzt wird, hat einen Wert von 1. . .128. Dieser Wert ist unter der Skizze angegeben. Um also den Wert zu ermitteln, der insgesamt in die jeweilige Speicherzelle geschrieben werden muß, muß man alle Werte der einzelnen Bits addieren. Wenn man z. B. eine Zeile komplett setzen will, müssen die Werte aller Bits addiert werden, da ja alle Punkte gesetzt werden sollen: Gesamtwert = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255.

```
>POKE &HF400, 255 : PRINT CHR$( 128 )
```

setzt die erste Zeile von ASCII-zeichen Nummer 128.

Zugegeben, dies klingt etwas kompliziert, aber anhand von einem weiteren Beispiel werden Sie sehen, daß das Definieren von Zeichen recht einfach ist, wenn man einmal das Prinzip verstanden hat.

Hier ist nun ein etwas aufwendigeres Beispiel für die Programmierung von Zeichen. Und zwar wollen wir das große „Ä“ definieren. Wenn man also mit Umlauten arbeiten will, muß man diese selbst definieren.

Ein „Ä“ sieht, nach Punkten aufgelöst, so aus:

	Bit-nummer	7	6	5	4	3	2	1	0
Adressen (hex.)	F400	■	x	x	■	x	x	■	x
(für ASCII-Zeichen	F401	x	x	■	x	■	x	x	x
Nummer 128)	F402	x	■	x	x	x	■	x	x
	F403	x	■	x	x	x	■	x	x
	F404	x	■	■	■	■	■	x	x
	F405	x	■	x	x	x	■	x	x
	F406	x	■	x	x	x	■	x	x
	F407	x	x	x	x	x	x	x	x
2 hoch Bitnummer = Wert des Bits		128	64	32	16	8	4	2	1

Nun müßten wir die Werte, die in die Speicherzellen von F400H bis F407H geschrieben werden sollen, berechnen. Dazu addieren wir in jeder Zeile die Werte der Punkte, die gesetzt werden sollen.

Berechnung:		Befehl:
Zeile 1:	128 + 16 + 2 = 146	>POKE &HF400, 146
Zeile 2:	32 + 8 = 40	>POKE &HF401, 40
Zeile 3:	64 + 4 = 68	>POKE &HF402, 68
Zeile 4:	64 + 4 = 68	>POKE &HF403, 68
Zeile 5:	64 + 32 + 16 + 8 + 4 = 124	>POKE &HF404, 124
Zeile 6:	64 + 4 = 68	>POKE &HF405, 68
Zeile 7:	64 + 4 = 68	>POKE &HF406, 68
Zeile 8:	0	>POKE &HF407, 0

Wenn Sie alle rechts aufgeführten „POKEs“ ausgeführt haben, ist das „Ä“ definiert!
Mit

```
>PRINT CHR $ ( 128 )
Ä
READY
>■
```

können Sie es sehen.

Genau nach obigem Muster können Sie beliebige Zeichen definieren. Wollten Sie nun ASCII-Zeichen Nummer 129 programmieren, müssen Sie nur die „POKE“-Adressen entsprechend ändern: Nr. 129 belegt die Speicherzellen von F408H bis F40FH, Zeichen Nr. 130 die Speicherzellen von F410H bis F417H usw.

35. Die Tonausgabe

Das Colour-Genie gibt den Ton über den Lautsprecher Ihres Fernsehers aus. Für die Tonausgabe gibt es im Genie-Basic 2 Befehle:

Der „PLAY"-Befehl

Allgemeines Format:

PLAY (Kanal, Oktave, Note, Lautstärke)

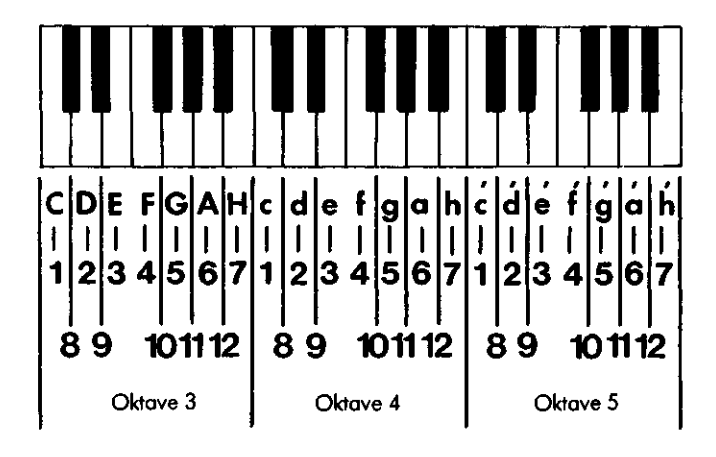
Für jedes der obigen Worte geben Sie eine Variable oder eine feste Zahl ein. Rechenausdrücke im „PLAY"-Befehl sind nicht zulässig! Dabei gilt folgendes Schema:

Kanal	1...3	Zur Auswahl eines Kanals. (Dadurch, daß 3 Kanäle zur Verfügung stehen, können auch Akkorde programmiert werden.)
Oktave	1...8	Zur Wahl der gewünschten Oktave.
Note	1...7	Für die Noten der C-dur-Tonleiter (entsprechend den weißen Tasten beim Klavier).
Note	8...12	Für die Halbtöne (entsprechend den schwarzen Tasten).
Lautstärke	0...15	Dieser Wert gibt die Lautstärke der jeweiligen Note an. Bei Lautstärke 0 ist nichts mehr zu hören — diesen Wert also für Pausen benutzen.

Ein Beispiel:

>PLAY (1,4,1,15)

gibt ein C in der 4. Oktave bei voller Lautstärke aus.



Das folgende Programm spielt ein bekanntes Stück:

```
>10 COLOUR 16 : CLS : PRINT @ 410, "Der Flohwalzer"
>20 R=RND ( 50 ) +30
>30 GOSUB 130 : GOSUB 130
>40 GOSUB 170
>50 L=2 : PLAY ( 1,2,10,15 ) : GOSUB 200 : GOSUB 180
>60 PLAY ( 1,2,9,15 ) : GOSUB 200 : GOSUB 180
>70 PLAY ( 1,2,8,15 ) : GOSUB 200 : GOSUB 150
>80 GOSUB 160 : GOSUB 160
>90 GOSUB 170 : PLAY ( 1,2,8,15) : L=2 : GOSUB 200 : L=2 : GOSUB 190
>100 PLAY ( 1,2,9,15) : GOSUB 200 : GOSUB 190
>110 GOSUB 140
>120 PLAY ( 1,1,1,0 ) : PRINT @ 920, ; : INPUT"Bitte <RETURN> drücken" ; A$ :
GOTO 10
>130 GOSUB 170
>140 L=2 : PLAY ( 1,2,10,15 ) : GOSUB 200 : GOSUB 180 : GOSUB 180 : RETURN
>150 L=2 : GOSUB 190 : GOSUB 190 : RETURN
>160 GOSUB 170 : PLAY ( 1,2,8,15 ) : L=2 : GOSUB 200 : GOSUB 150 : RETURN
>170 L=1 : PLAY ( 1,3,9,15 ): GOSUB 200 : PLAY ( 1,3,8,15 ) : GOSUB 200 : RETURN
>180 PLAY ( 1,3,12,15 ) : PLAY ( 2,4,10,15 ) : L=2 : GOSUB 200 : PLAY ( 2,1,1,0 ) :
RETURN
>190 PLAY ( 1,3,7,15 ) : PLAY ( 2,4,4,14 ) : L=2 : GOSUB 200 : PLAY ( 2,1,1,0 ) :
RETURN
>200 FOR A1=1 TO L★R : NEXT : RETURN
```

Der „SOUND"-Befehl

Allgemeines Format:

```
>SOUND R,N
```

mit R=0...15 und N=0...255.

Mit diesem Befehl wird der Wert N direkt in das Register R des PSG (Programmable Sound Generator) geladen.

Eine Auflistung der PSG-Register finden Sie im Anhang.

Ein Programmbeispiel zum „SOUND"-Befehl:

```
>10 CLS : PRINT "SCHUSS-GERAEUSCH"
>20 FOR I = 1 TO 11
>30 SOUND 6,30 : SOUND 7,7 : SOUND 8,16 : SOUND 9,16
>40 SOUND 10,16 : SOUND 12,16 : SOUND 13,0
>50 FOR B = 1 TO 150 : NEXT
>60 COLOUR A : PRINT"PENG",
>70 NEXT
```

In Anhang B finden Sie eine genaue Beschreibung, wie der PSG programmiert wird.

Ferner ist in Anhang K ein Programm aufgelistet, das in Form einer Bildschirmmaske ein sehr komfortables Hilfsmittel zum Experimentieren darstellt.

36. Steuerknüppel und externe Tastaturen

Das Colour-Genie bietet die Möglichkeit, zwei Steuerknüppel (analog Joysticks) und zwei externe Tastaturen (external Keypads) anzuschließen. Diese können als Zusatzgeräte erworben werden. Das Basic des Colour-Genies unterstützt Steuerknüppel und externe Tastaturen mit folgenden Befehlen:

JOY1X

Gibt die X-Koordinate des 1. Steuerknüppels an.
Z. B.:
>PRINT JOY1X

JOY1Y

Gibt die Y-Koordinate des 1. Steuerknüppels an.
Z. B.:
>PRINT JOY1Y

JOY2X

Gibt die X-Koordinate des 2. Steuerknüppels an.
Z. B.:
>X=JOY2X

JOY2Y

Gibt die Y-Koordinate des 2. Steuerknüppels an.
Z. B.:
>Y=JOY2Y

Der Bereich der Steuerknüppel-Koordinaten liegt von 1. . .64.

KEYPAD1

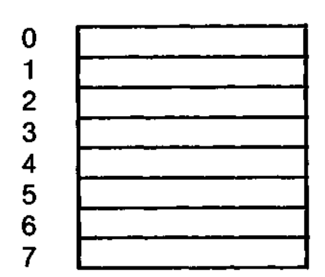
Ergibt den Wert einer gedrückten Taste der 1. Tastatur.
Z. B.:
>PRINT KEYPAD1

KEYPAD2

Ergibt den Wert einer gedrückten Taste der 2. Tastatur.
Z. B.:
>PRINT KEYPAD2

37. Programmierbarer Cursor

Beim Einschalten ist der Cursor des Colour-Genies auf schnell blinkend und volle Größe eingestellt. Man kann jedoch Größe und Blinkfrequenz programmieren und den Cursor auch nach folgendem Schema ganz abschalten:




```
POKE 16409, N1
POKE 16410, N2 + N3
```

Dabei haben N1, N2 und N3 folgende Bedeutung:

N1: Gibt die Nummer der untersten Zeile des Cursors gemäß der obigen Skizze an. Werte von 0. . .7 sind zulässig.

N2: Gibt die Nummer der obersten Zeile des Cursors an. Auch hierfür ist ein Wert von 0. . .7 einzusetzen.

N3: N3 darf 4 verschiedene Werte annehmen:

N3 = 0:	Cursor blinkt nicht
N3 = 32:	kein Cursor
N3 = 64:	schnell blinkender Cursor
N3 = 96:	langsam blinkender Cursor

Dazu ein Beispiel:

Sie wollen einen Cursor haben, der sich nur auf Cursorzeile 5, 6 und 7 erstrecken soll, also die unteren drei Cursorzeilen (siehe obige Skizze). Dabei soll er langsam blinken. Dies wird durch folgende „POKEs" erreicht:

```
>POKE 16409,7
READY
>POKE 16410,101                ( 101 = 5 + 96 )
READY
>■
```

Achten Sie bitte, wie bei allen „POKE"-Befehlen, darauf, daß Sie die POKE-Adressen richtig eingeben — ein falsches „POKE" kann nämlich katastrophale Folgen für Ihr Programm haben.

ANHANG

A. Der CRTC

Hinweis:
Dieses Kapitel ist für den Programmier-Anfänger wahrscheinlich zu schwierig. Für das Verständnis des Genie-Basic ist es aber auch nicht nötig.

CRTC ist eine Abkürzung für *C*athode *R*ay *T*ube *C*ontroller. Darunter hat man ein hochintegriertes Bauteil zu verstehen, das sämtliche Bildschirmausgaben des Colour-Genies organisiert. Dabei sind sogar Werte wie das Bildschirmformat oder die Synchronisation des Fernsehers frei einstellbar.

Im Colour-Genie Basic sind die Werte des CRTCs auf folgende Adressen gelegt:

Für Text-Modus:

Adresse (dezimal)	Pal-Wert beim Einschalten	Inhalt, Kommentar
17136 17137	1 0	Cursor-Adresse Siehe Anm. 1
17138 17139	0 4	Bildschirmspeicher Startadresse = 4400H
17140 17141	7 196	Cursor End Raster Cursor Blinken, Cursor Start Raster. (Siehe Anm. 1)
17142	7	Zeichen-End-Raster
17143	160	Interlace and Skew
17144	30	Vertikale Synchronisationsposition. Ermöglicht es, das Bild vertikal zu versetzen.
17145	24	Zeilenzahl
17146	0	Vertikale Justierung — Gegenstück zum Bildfang beim Fernseher.
17147	38	Vertikale Gesamtzahl der auf dem Fernseher angezeigten Elektronenstrahl-Linien.
17148	150	Länge des Zeilensynchronpulses
17149	52	Horizontale Synchronisationsposition. Ermöglicht das Bild horizontal zu versetzen.
17150	40·	Zeichen pro Zeile
17151	70	Horizontale Abtastgeschwindigkeit

Wird auf hochauflösende Grafik umgeschaltet, so wird der CRTC mit neuen Werten programmiert. Das ist auch der Grund dafür, daß Grafik und Text nicht gemischt werden können.

Werte für Grafik-Modus:

Adresse (dezimal)	Pal-Wert beim Einschalten	Inhalt, Kommentar
17152 17153	0 0	Cursor-Adresse (Siehe Anm. 1)
17154 17155	0 8	Bildschirmspeicher Startadresse = 4800H
17156 17157	0 32	Cursor End Raster Cursor Blinken, Cursor Start Raster. (Siehe Anm. 1)
17158	1	Zeichen-End-Raster
17159	32	Interlace and Skew
17160	108	Vertikale Synchronisationsposition. Ermöglich es, das Bild vertikal zu versetzen.
17161	96	Zeilenzahl
17162	31	Vertikale Justierung
17163	126	Vertikale Gesamtzahl
17164	150	Länge des Zeilensynchronpulses
17165	52	Horizontale Synchronisationsposition
17166	40	Zeichen pro Zeile
17167	70	Horizontale Abtastgeschwindigkeit

Anm. 1: Diese Werte sind im Genie-Basic irrelevant, da das Basic seinen eigenen Cursor programmiert.

Die oben aufgelisteten Adressen gelten nur für den Betrieb im Basic. Direkt adressiert werden kann der CRTC über die beiden Ports 250 und 251 (dez.). Dabei wird über Port 250 die Registernummer des CRTC ausgegeben und über Port 251 der jeweilige Wert. (Register 0 des CRTC entspricht Basic-Adresse 17151 bzw. 17167 im Grafikmodus. Register 15 der Adresse 17136 bzw. 17152. Genaue Beschreibungen der CRTC-Register entnehmen Sie den Applikationsschriften der Hersteller. Die Typenbezeichnung des CRTCs ist 46505S.)

Abschließend noch ein Programmzeile, die zeigt, wie man die Bildschirmausgabe vom Basic her steuern kann:

```
>10 FORB=1TO10 : FORA=52TO10STEP—1 : POKE17149,A : LGR : NEXT :
FORA=10TO52 : POKE17149,A : LGR : NEXT : NEXT
```

Beachten Sie, daß nach jedem „POKE" der aktive Modus — in diesem Falle „LGR" — neu aufgerufen werden muß.

B. Der PSG

Hinweis: Dieses Kapitel ist für den Programmier-Anfänger wahrscheinlich zu schwierig. Für das Verständnis des Genie-Basic ist es aber auch nicht nötig.

PSG ist eine Abkürzung für *P*rogrammable *S*ound *G*enerator. Dies ist, wie der CRTC, ein hochintegriertes Bauteil, das die gesamte Tonausgabe des Colour-Genies übernimmt. Ausgegeben wird der Ton über den Lautsprecher des Fernsehers. Dabei können bis zu 3 Tonkanäle mit Frequenz, Hüllkurve und Rauschen belegt werden.

Programmiert wird der PSG, ähnlich dem CRTC, durch das Speichern von Werten in die PSG-Register.

Dies ist, wie im Kapitel „Die Tonausgabe" beschrieben, direkt mit dem Befehl „SOUND R,N" möglich, wobei der Wert N (0...255) in das Register R (0...15) übernommen wird.

Es ist ebenfalls möglich, den PSG über die beiden Ports 248 und 249 (dez.) zu aktivieren.

"OUT 248,R : OUT 249,N" entspricht also "SOUND R,N".

Die PSG-Register:

Register	Funktion	Einstellung
R0	Kanal A Tonfrequenz	Feineinstellung (0...255)
R1		Grobeinstellung (0...15)
R2	Kanal B Tonfrequenz	Feineinstellung (0...255)
R3		Grobeinstellung (0...15)
R4	Kanal C Tonfrequenz	Feineinstellung (0...255)
R5		Grobeinstellung (0...15)
R6	Rauschfrequenz	Einstellung (0...31)
R7	Aktivierung	Siehe Tabelle 1 (0—255)
R8	Kanal A Lautstärke	Aus...Laut (0...15) 16: moduliert mit Hüllkurve
R9	Kanal B Lautstärke	Aus...Laut (0...15) 16: moduliert mit Hüllkurve
R10	Kanal C Lautstärke	Aus...Laut (0...15) 16: moduliert mit Hüllkurve
R11	Hüllkurve Frequenz	Feineinstellung (0...255)
R12		Grobeinstellung (0...255)
R13	Hüllkurvenform	Siehe Tabelle 2 (0...15)
R14	Parallelport 1	(Druckerport, Joystick ...)
R15	Parallelport 2	(Druckerport, Joystick ...)

Register 7 Aktivierung der Kanäle für Töne und Rauschen und festlegen ob Eingabe- oder Ausgabe-Ports.

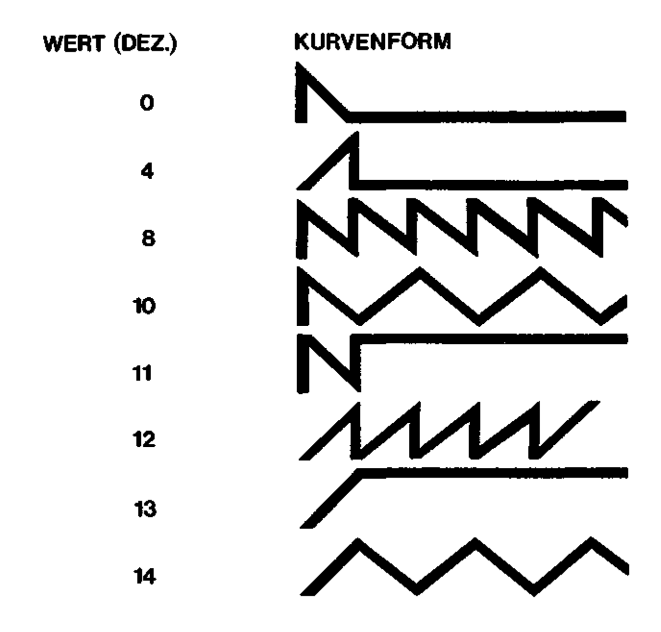
Bit 0	Ton Kanal A	Wenn gesetzt, *kein* Ton.
Bit 1	Ton Kanal B	Wenn gesetzt, *kein* Ton.
Bit 2	Ton Kanal C	Wenn gesetzt, *kein* Ton.
Bit 3	Rauschen Kanal A	Wenn gesetzt, *kein* Rauschen.
Bit 4	Rauschen Kanal B	Wenn gesetzt, *kein* Rauschen.
Bit 5	Rauschen Kanal C	Wenn gesetzt, *kein* Rauschen.
Bit 6	I/O Port 1	Wenn gesetzt, Ausgabe.
Bit 7	I/O Port 2	Wenn rückgesetzt, Eingabe.

Beispiel:

Angenommen, wir wollen über Kanal A einen Ton, über Kanal B Ton und Rauschen und über Kanal C nichts ausgeben sowie Port 1 als Eingabe-Port benutzen und Port 2 zur Ausgabe verwenden:

Wir schreiben in Register 7 also:

Bit 2, Bit 3, Bit 5, Bit 7; alle anderen Bits sind zurückgesetzt: 4+8+32+128=172, also "SOUND 7,172".



Ein Beispiel für die Programmierung des PSG:

Folgendes Programm erzeugt so etwas Ähnliches wie Dampflokgeräusche:

```
>10 SOUND 6,10 : REM  Rauschfrequenz einstellen
>20 SOUND 7,247 : REM  Rauschen Kanal A aktivieren
>30 SOUND 8,16 : REM  Amplitude auf Hüllkurve setzen
>40 SOUND 12,25 : REM  Hüllkurvenfrequenz festlegen
>50 SOUND 13,8 : REM  Art der Hüllkurve festlegen
>60 FOR =1 TO 1000 : NEXT
>70 PLAY ( 2,5,3,15 ) : REM 2=Kanal B aktiviert
>80 FOR A=1 TO 500 : NEXT
>90 PLAY ( 2,4,1,0 )
>100 SOUND 7,247 : REM  Kanal B abstellen
>110 SOUND 8,15 : REM  Amplitude auf konstant volle Lautstärke
>120 FOR A=1 TO 23
>130 FOR B=1 TO 200 : NEXT
>140 SOUND 8,16—A / 1.5 : REM  Lautstärke herabsetzen
>150 NEXT
```

An dieser Stelle wird nochmal auf Anhang K hingewiesen, in dem ein Programm zur Programmierung des PSG aufgelistet ist.

C. Control Codes

Mit „PRINT CHR$(X)“ und Werten von X<32 können diverse Steuerfunktionen ausgeführt werden, wobei nicht alle Werte von 0...31 belegt sind, sondern nur die hier aufgelisteten:

Code	Funktion
8	Löschen des letzten Zeichens, Cursor ein Zeichen nach links.
10—13	Carriage Return (Cursor zum Anfang der nächsten Zeile)
14	Cursor anschalten
15	Cursor ausschalten
24	Backspace cursor (Cursor eine Position nach links)
25	Advance cursor (Cursor eine Position nach rechts)
26	Downward linefeed (Cursor eine Zeile tiefer)
27	Upward linefeed (Cursor eine Zeile höher)
28	Home cursor (Cursor auf Bildschirmposition 0)
29	Cursor zum Zeilenanfang
30	Löschen bis zum Ende der Zeile
31	Löschen bis zum Ende des Bildschirms

Mit folgendem kleinen Programm können Sie die Steuercodes ausprobieren:

```
>10 CLS
>20 INPUT "Eingabe des Steuercodes ( 8... 31 )" ; X
>30 PRINT"ABC"CHR$ ( X ) ; " DEF"
>40 GOTO 20
```

D Speicherbereiche und Grenzen der Programmierung

Bereiche:

ganze Zahlen	—32768 bis +32767 einschl.
einfache Genauigkeit	—1.701411E + 38 bis +1.701411E + 38 einschl.
doppelte Genauigkeit	—1.701411834244556 D + 38 bis +1.701411834244556 D + 38 einschl.
String Bereich	bis 255 Zeichen
Zeilennummern	0 Bis 65529 einschl.
Länge der Programmzeilen	bis zu 255 Zeichen

Speicherplatzbedarf
Eine Programmzeile braucht minimal 5 Bytes:

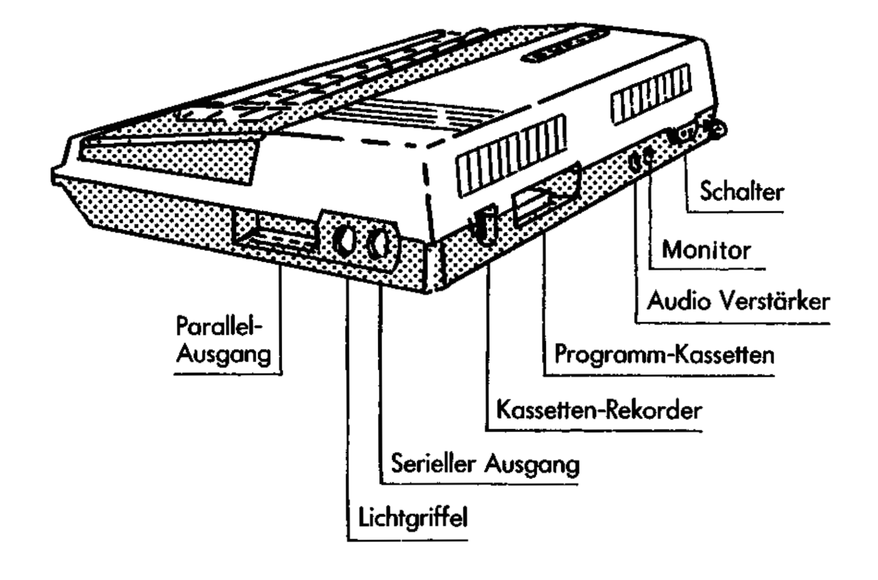
Zeilenummer:	2 Bytes
Zeilenpointer:	2 Bytes
Carriage Return (Zeilenvorschub):	1 Byte

Dazu braucht jeder Befehl, Operator, Variablenname, Spezialzeichen und Konstantenziffer ein Byte.

Dynamische Speicherplatzzuteilung (bei Programmausführung)

ganzzahlige Variablen	5 Bytes jede
Variablen einfacher Genauigkeit	7 Bytes jede
Variablen doppelter Genauigkeit	11 Bytes jede
String Variablen	6 Bytes minimal (3 für Variablennamen, 3 für Länge und Pointer, 1 für jedes Zeichen)
Feldvariablen	12 Bytes minimal (3 für Variablennamen, 2 für Größe, 1 für Anzahl der Dimensionen und 2, 3, 4 oder 8 für jedes Element, abhängig vom Typ)
Jede aktive FOR-NEXT-Schleife	16 Bytes
Jedes aktive GOSUB (noch kein Return)	6 Bytes
Jede Klammerebene	4 Bytes und 12 Bytes für Zwischenergebnisse

E. Anschlußmöglichkeiten beim Colour-Genie



F. ASCII Code

hex	dez.	ASCII	hex	dez.	ASCII	hex	dez.	ASCII	hex	dez.	ASCII
00	0	NUL	20	32	Space	40	64	@	60	96	’
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	—	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6C	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	‘	7C	124	’
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	Λ	7E	126	~
1F	31	US	3F	63	?	5F	95	—	7F	127	DEL

G. Speicherbelegung

Adresse	Belegung
0000 bis 3FFF	16K BASIC ROM
4000 bis 43FF	Kommunikationsbereich (CRTC, Kassetten-Status, Disk-Status . . .)
4400 bis 47FF	Bildschirmspeicher Textmodus
4800 bis 57FF	Bildschirmspeicher für hochauflösende Grafik
5800 bis 7FFF	Benutzer RAM
8000 bis BFFF	Weitere 16K Benutzer RAM (nachrüstbar im Gerät)
C000 bis EFFF	ROM-Einschubkassette (Cartridge)
F000 bis F3FF	Speicher für die Zeichenfarbe im Textmodus
F400 bis F7FF	Speicher für programmierte Zeichen
F800 bis FBFF	Tastatur
FC00 bis FFFF	ROM-Einschubkassette (Cartridge)

In diesem Zusammenhang noch die Erklärung der „MEM SIZE?" Frage, die der Computer beim Einschalten stellt. Als Antwort können Sie hier eine 5stellige Dezimalzahl eingeben. Wenn Sie einen korrekten Wert eingeben, sind alle Speicherzellen oberhalb dieses Wertes reserviert, d. h. sie werden vom Basic nicht mehr benutzt. Dadurch stehen Sie dann z.B. für Maschinenspracheroutinen zur Verfügung.

H. Tastatur-Speicher

Beim Colour-Genie ist die Tastatur in den Speicher gelegt, d. h. jede Taste belegt ein Bit im Speicher:

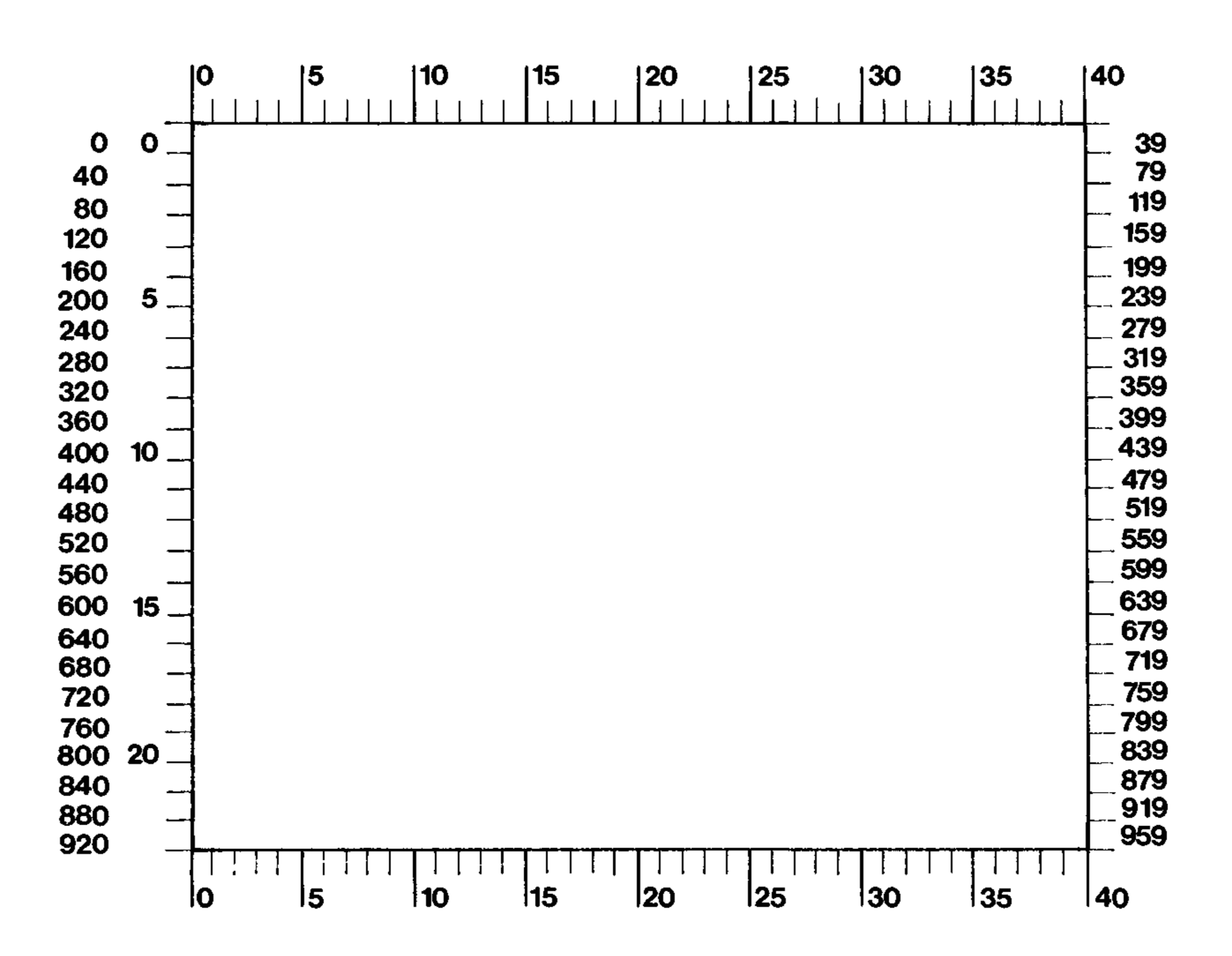
ADRESSE (HEX)	BIT NR.	0	1	2	3	4	5	6	7
	F801	@	A	B	C	D	E	F	G
	F802	H	I	J	K	L	M	N	O
	F804	P	Q	R	S	T	U	V	W
	F808	X	Y	Z		F1	F2	F3	F4
	F810	0	1	2	3	4	5	6	7
	F820	8	9	:	;	,	–	.	/
	F840	RET.	CLR	BRK	↑	↓	←	→	
	F880	SHIFT	M.S.		RPT	CTRL			

Sie können also Tasten auch mit dem „PEEK“-Befehl abfragen. Z. B.:

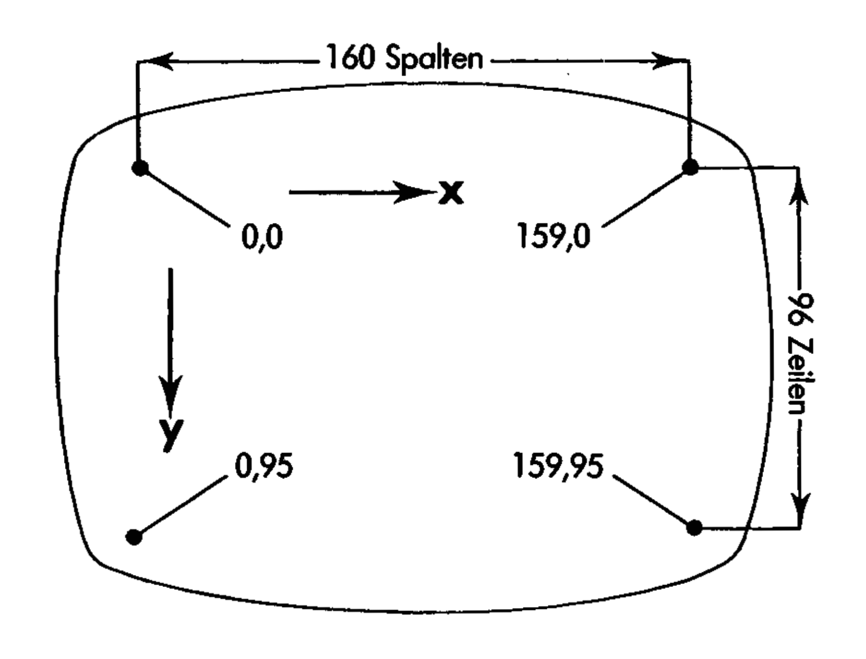
```
>10 PRINT PEEK ( &HF889 ) : GOTO 10
>RUN
```

Wenn Sie jetzt die <SHIFT>, <MOD SEL>, <RPT> oder <CRTL>-Taste drücken, wird der entsprechende Wert angezeigt.

I. Die PRINT @ -Positionen des Colour-Genies (LGR-Modus)



J. Die Grafik-Koordinaten des Colour-Genies (HGR-Modus)



Farben:

FCOLOUR 1 = schwarz
FCOLOUR 2 = blau
FCOLOUR 3 = orange
FCOLOUR 4 = grün

K. Programme

Auf den folgenden Seiten finden Sie die Listungs folgender vier Programme:

1.) „CODE BREAKER“
Dies ist ein relativ einfaches Denkspiel, bei dem es darum geht, eine 4stellige Zahlenkombination herauszufinden.
Das Programm erklärt sich selbst.

2.) „SABOTAGE“
Sabotage ist ein Grafikspiel, bei dem die definierbaren Zeichen Ihres Colour-Genies auf ansprechende Weise benutzt werden. Auch dieses Spiel ist selbsterklärend.

3.) „SOUND EDITOR“
Mit Hilfe dieses Programms können Sie leicht alle Möglichkeiten des PSGs erschließen. Dabei werden sämtliche Register bitweise auf dem Bildschirm editiert.
Eine Beschreibung der Funktion der einzelnen PSG-Register finden Sie in Anhang B.
Die genaue Bedienung des „Sound-Editors“ wird durch das Programm selbst erklärt.

4.) „EDITOR FÜR DEFINIERBARE ZEICHEN“
Wie bereits in Kapitel 33 beschrieben ist, kann man beim Colour-Genie bis zu 128 Zeichen frei definieren. Da diese Programmierung gewöhnlich Zeichen für Zeichen und obendrein noch bitweise geschieht, ist das Ganze recht aufwendig.
Mit Hilfe dieses Programms können Sie nun bis zu 64 Zeichen gleichzeitig auf dem Bildschirm bequem editieren. Abschließend bietet das Programm sogar die Möglichkeit, von sich aus ein neues Programm zu erstellen, das die entworfenen Zeichen in „DATA“-Zeilen abspeichert.

Nun noch einige allgemeine Worte zur Handhabung der auf den nächsten Seiten folgenden Listungs:

Alle vier Programme sind von routinierten Programmierern extra für dieses Handbuch geschrieben.

Daher enthalten diese Programme eine Vielzahl von Programmtricks. Seien Sie also nicht verärgert, wenn Sie nicht alle Programmschritte verstehen. Vorsichtig sollten Sie beim Abtippen der Programme sein, eine falsche Zahl kann Ihnen z. B. eine Menge Schwierigkeiten bereiten. Die beste Lösung ist, wenn Ihnen jemand die Programmzeilen diktiert, der möglichst gleichzeitig Ihre Eingaben kontrolliert. Sollten Sie aber die Programme allein eingeben, kontrollieren Sie am besten jede Zeile, nachdem Sie sie eingegeben haben. Dies kostet zwar etwas Zeit, kann Ihnen aber eine schwierige Fehlersuche ersparen.

(In den Listings entspricht § der <@>–Taste, Ä der <↑>–Taste)

1. Code Breaker

```
10 CLS:RANDOM:DEFINTA-Z
20 COLOUR9
30 PRINT"              *** CODE BREAKER ***"
40 PRINTSTRING$(40,217);
50 FORA=1TO8:FORB=1TO12:PLAY(1,A,B,11):NEXT:NEXT:PLAY(1,1,1,0)
60 COLOUR16
70 PRINT"Sie sollen einen 4-stelligen Code
80 PRINT"herausfinden. Dieser Code setzt sich
90 PRINT"aus 4 Zahlen zusammen. Jede Zahl
100 PRINT"kann eine Wert von 1...9 annehmen.
110 PRINT"Dabei kommen keine Zahlen doppelt vor.
120 PRINT
130 PRINT"Nach jeder Eingabe gibt der Computer an,";
140 PRINT"wieviele Zahlen richtig sind, aber an
150 PRINT"der falschen Stelle stehen (TREFFER),
160 PRINT"und wieviele Zahlen richtig sind,
170 PRINT"die auch an der richtigen Stelle
180 PRINT"stehen (VOLLTREFFER).
190 PRINT"Das Spiel ist beendet, wenn Sie 4
200 PRINT"Volltreffer hatten."
210 PRINT
220 PRINT"Zur Eingabe Ihres 4-stelligen Versuchs
230 PRINT"geben Sie einfach die 4 Zahlen ein
240 PRINT"und druecken anschliessend <RETURN>.
250 PRINT"Korrekturen wie ueblich mit <"CHR$(253)">."
260 COLOUR9:PRINTSTRING$(40,217);
270 A$=INKEY$
280 COLOURRND(16)
290 PRINT§920,"Zum Start <RETURN> druecken !";
300 PLAY(1,4,5,11)
310 FORA=0TO200:NEXT
320 PLAY(1,4,5,0)
330 PRINT§920,CHR$(31);
340 FORA=0TO200:NEXT
350 I$=INKEY$:IFI$=""THEN270
360 IFASC(I$)=13THENCLSELSE270
370 COLOUR16
380 PRINT§0,"Nr.  Eingabe    Volltreffer    Treffer"
390 PRINTSTRING$(40,208);
400 F=0:FORA=1TO4
410 A(A)=RND(9)
420 FORB=A-1TO0STEP-1:IFA(A)=A(B)THENF=1:NEXT:GOTO410ELSENEXT
430 IFF=1THENF=0:GOTO410
440 NEXT
450 N=1
460 PRINTUSING"##";N;:PRINTCHR$(32)CHR$(225)CHR$(32);:PRINTSTRING$(9,32)CHR$(225
)STRING$(14,32)CHR$(225);STRING$(25,24);
470 FORA=5TO8:PRINTCHR$(14);:GOSUB740:A(A)=V:NEXT
480 V=0:T=0:N=N+1
490 PLAY(1,1,7,15)
500 FORA=1TO4:B(A)=A(A+4):NEXT
510 FORA=1TO4
520 P=8-A*2:PLAY(1,1,P,15)
530 IFA(A)=B(A)THENV=V+1:FORB=1TO4ELSENEXT:GOTO560
540 IFA(A)=B(B)THENB(B)=0
550 NEXT:NEXT
560 PLAY(1,1,1,0):FORA=1TO4:FORB=1TO4:IFA=BTHEN610
570 IFB(B)=0THEN610
580 IFB(B)=A(A)THENT=T+1:FORC=1TO4ELSE610
590 IFA(A)=B(C)THENB(C)=0
600 NEXT
610 NEXT:NEXT
```

```
620 PRINTSTRING$(5,25);
630 IFV=0THEN640ELSEFORA=1TOV:PRINT"* ";:NEXT
640 FORA=VTO4:PRINT"  ";:NEXT
650 PRINTSTRING$(3,25);
660 IFT=0THEN670ELSEFORA=1TOT:PRINT"+ ";:NEXT
670 PRINT
680 IFV<>4THEN460
690 FORA=1TO10:BGRD:FORB=1TO30:NEXT:NBGRD:FORB=1TO30:NEXT:NEXT:COLOUR5:PRINT:PRI
NT"Sie brauchten"N-1"Versuche, Glueckwunsch "
700 PRINT"<RETURN> fuer ein neues Spiel '"
710 IFINKEY$=""THEN710
720 CLS:GOTO370
730 GOTO460
740 A$=INKEY$:V=VAL(A$):IF(A$=CHR$(31)ORA$=CHR$(8))ANDA>5THENA=A-1:GOTO790ELSEIF
V=0THEN740
750 P=A-4:Q=5+A:PLAY(1,4,P,Q):FORP=1TO100:NEXT:PLAY(1,4,1,0)
760 IFA=8THENPRINTCHR$(V+48)CHR$(32);:GOTO810
770 PRINTCHR$(V+48)CHR$(32)CHR$(15);
780 RETURN
790 GOSUB820:PRINTCHR$(8);CHR$(8);"  ";CHR$(8)CHR$(8);
800 GOSUB740:RETURN
810 A$=INKEY$:IFA$=CHR$(31)ORA$=CHR$(8)THEN790ELSEIFA$=CHR$(13)THENPRINTCHR$(15)
;:RETURNELSE810
820 SOUND6,30:SOUND7,7:SOUND8,16:SOUND9,16:SOUND10,16:SOUND12,16:SOUND13,0:RETUR
N
```

2. Sabotage

```
10 COLOUR16
20 CLS
30 CLEAR100:DEFINTA-Z:DEFSNGU,Z
40 PRINT"MACH' DICH BEREIT FUER......"
50 GOSUB1000
60 BGRD:FORT=2TO350:NBGRD
70 CLS
80 REM CARSTEN SCHMIDT & KALLE BRAUN
90 SOUND1,8:SOUND3,9:SOUND5,1:SOUND6,2:SOUND7,7:SOUND8,31:SOUND9,31:SOUND10,24:S
OUND12,80:SOUND13,3
100 PRINTTAB(16)"Sabotage"
110 PRINTTAB(16) STRING$(8,211)
120 PRINT§887,"Druecke (A) fuer Anleitung,
               (S) fuer Start.";
130 A$=INKEY$:IFA$=""THENBGRD:NBGRD:GOTO130
140 IFA$<>"S"ANDA$<>"A"THEN130
150 IFA$="A"GOSUB1090
160 CLS:IFA$="S"GOSUB1120
170 CLS
180 PRINT§15,"ZEIT:";
190 K=0:A$=CHR$(128):B$=CHR$(144)+" "+CHR$(146)+CHR$(147)+" "+CHR$(149)+CHR$(150
):C$="":FORT=160TO167:C$=C$+CHR$(T):NEXTT
200 COLOUR16
210 IFI>=0THENPRINT§I*40+J," ";
220 PRINT§856,A$;:PRINT§896,B$;:PRINT§936,C$;
230 COLOUR10:PRINT§927,CHR$(140);:COLOUR15:PRINT§935,CHR$(176);:PRINT§944,CHR$(1
76);:COLOUR10:PRINT§952,CHR$(140);
240 COLOUR16:PRINT§920,CHR$(172);CHR$(175);CHR$(174);CHR$(173);
250 COLOUR2
260 FORX=924TO926:PRINT§X,CHR$(175);:NEXTX:FORX=928TO934:PRINT§X,CHR$(175);:NEXT
X:FORX=945TO951:PRINT§X,CHR$(175);:NEXTX:FORX=953TO958:COLOUR11:PRINT§X,CHR$(140
+INT(RND(3)/2));:NEXTX
270 COLOUR6
280 H=RND(17):T=RND(TT):S=1:IFRND(2)=2 THEN S=-1
290 T=H:U=(23-H)/400
300 IF S=1 THEN Q=1 ELSE Q=38
310 FORX=QTO20STEPS:IF X>20THENZ=X-40ELSEZ=X
320 P=153+S*16:V=INT(U*ZA2+T):PRINT§V*40+X,CHR$(P);:PRINT§V*40+X+S,CHR$(P+S);
330 FOR Y=1 TO T:NEXT Y:PRINT§V*40+X," ";:PRINT§V*40+X+S," ";
340 ZZ=ZZ-.2:IFZZ<0GOTO1170ELSEPRINT§20,INT(ZZ);
350 IF K>0 THEN 400
360 IFPEEK(-1984)=32THENK=1ELSEIFPEEK(-1984)=64THENK=2ELSEK=0
370 ONKGOSUB440,450
380 IF K=0 THEN NEXTX:GOTO1030
390 SOUND1,6:SOUND3,5:SOUND5,11:SOUND6,2:SOUND7,7:SOUND8,31:SOUND9,31:SOUND10,24
:SOUND12,80:SOUND13,3
400 PRINT§I*40+J," ";
410 I=I-1:IFI<0THEN K=0:NEXTX:GOTO1040
420 IF((I>V-Z2)*(I<V+Z2))*((J=X)+(J=X+S))THENPRINT§I*40+J," ";:GOSUB1010:GOTO190
430 PRINT§I*40+J,CHR$(177);:NEXTX:GOTO1030
440 J=15:I=22:RETURN
450 J=24:I=22:RETURN
460 DATA048,048,048,048,048,048,048,048
470 DATA000,000,000,000,000,000,000,000
480 DATA000,000,000,000,000,000,000,000
490 DATA000,000,000,000,000,000,000,000
500 DATA000,000,000,000,000,000,000,000
510 DATA000,000,000,000,000,000,000,000
520 DATA000,000,000,000,000,000,000,000
530 DATA000,000,000,000,000,000,000,000
540 DATA000,000,002,031,031,002,000,000
```

```
550 DATA000,003,007,254,254,007,003,000
560 DATA000,000,000,000,000,000,000,000
570 DATA000,000,000,000,000,000,000,000
580 DATA000,000,000,000,024,062,087,171
590 DATA000,000,000,000,000,000,206,247
600 DATA000,000,000,000,000,000,000,000
610 DATA000,000,000,000,000,000,000,000
620 DATA048,048,048,048,048,048,048,048
630 DATA000,000,000,000,000,000,000,000
640 DATA000,000,002,015,031,055,127,255
650 DATA000,000,192,224,240,216,252,252
660 DATA000,000,000,000,000,000,000,000
670 DATA000,013,007,003,003,003,003,003
680 DATA000,214,252,120,232,248,104,248
690 DATA000,000,000,000,000,000,000,000
700 DATA000,000,000,000,000,000,000,000
710 DATA000,000,000,000,000,000,000,000
720 DATA000,000,000,000,000,000,000,000
730 DATA000,000,000,000,000,000,000,000
740 DATA000,000,000,000,000,000,000,000
750 DATA000,000,000,000,000,000,000,000
760 DATA000,000,000,000,000,000,000,000
770 DATA000,000,000,000,000,000,000,000
780 DATA048,048,048,048,056,054,127,255
790 DATA000,001,001,003,003,007,255,175
800 DATA255,255,255,247,255,255,255,223
810 DATA254,254,255,239,255,255,252,255
820 DATA000,000,000,000,128,127,170,255
830 DATA003,003,003,007,023,107,183,255
840 DATA248,248,248,252,254,251,255,221
850 DATA000,000,000,000,000,128,224,184
860 DATA000,000,000,000,000,000,000,000
870 DATA000,192,224,127,127,224,192,000
880 DATA000,000,064,248,248,064,000,000
890 DATA000,000,000,000,000,000,000,000
900 DATA000,000,000,000,119,093,119,255
910 DATA000,000,000,000,000,254,149,255
920 DATA032,032,032,080,080,127,119,253
930 DATA000,000,000,000,000,000,000,255
940 DATA000,036,036,036,036,036,102,189
950 DATA000,000,024,024,024,024,024,024
960 DATA000,000,000,000,000,000,000,000
970 DATA000,000,000,000,000,000,000,000
980 DATA000,000,000,000,000,000,000,000
990 DATA000,000,000,000,000,000,000,000
1000 CHAR2:RESTORE:FORA%=&HF400TO&HF5AF::READB%:POKEA%,B%:NEXT:RETURN
1010 COLOUR16:SOUND0,0:SOUND1,8:SOUND2,0:SOUND3,8:SOUND4,0:SOUND5,8:SOUND6,24:SO
UND7,7:SOUND8,24:SOUND9,24:SOUND10,0:SOUND11,48:SOUND12,45:SOUND13,3:PRINT§V*40+
X,CHR$(230);:FORT=1TO10:NEXTT:PRINT§V*40+X," ";:COLOUR6:RETURN
1020 COLOUR5
1030 FORA=0TO255:OUT255,A:NEXT:FORB=1TO3:FORA=52TO42STEP-1:POKE17149,A:LGR:NEXT:
FORA=42TO52:POKE17149,A:LGR:NEXT:NEXT
1040 PRINT§920,STRING$(39,230);:FORT=1TO40:NEXTT:BGRD:FORT=1TO10:NEXTT:PRINT§880
,STRING$(39,RND(50)+200);:NBGRD:SOUND0,255:SOUND1,15:SOUND2,255:SOUND3,15:SOUND4
,255:SOUND5,15:SOUND6,23:SOUND7,7:SOUND8,31:SOUND9,31:SOUND10,31:SOUND11,255:SOU
ND12,255:SOUND13,9
1050 COLOUR16
1060 CLS:FORT=1TO2000:NEXTT
1070 CLS:PRINT"HAST DU EIN SCHWEIN, DASS ALLES
NUR SPIEL IST...."
1080 PRINT:PRINT"NOCHMAL?":GOTO1180   x (Y/N)
1090 CLS:PRINT"EIN KKW (TYP 'SCHNELLER BRUETER')
WIRD VON SABOTEUREN (TYP 'UNBEKANNT')
```

```
1095 PRINT"MIT RAKETEN BESCHOSSEN. DIESE RAKETEN
MUESSEN MIT DEN TASTEN ";CHR$(253);" UND ";CHR$(255);" ABGE-
SCHOSSEN WERDEN, SONST GIBT ES EINE
FURCHTBARE ATOMKATASTROPHE.
1100 PRINT"BEACHTE, DASS NUR IMMER EIN SCHUSS
ABGEFEUERT WERDEN DARF!!"
1110 PRINT:PRINT"DAS SPIEL IST ZU ENDE, WENN DU DAS KKW
DIE ANGEGEBENE ZEITLANG VERTEIDIGEN
KONNTEST.

BITTE WAEHLE DEINEN SCHWIERIGKEITSGRAD:"
1120 PRINT:PRINT"(1) --- ANFAENGER":PRINT"(2) --- FORTGESCHRITTENER":PRINT"(3) -
-- PROFI"
1130 A$=INKEY$:IFA$=""THEN1130
1140 IFVAL(A$)<1ORVAL(A$)>3THEN1130
1150 IFVAL(A$)=1THENTT=50:IFVAL(A$)=2THENTT=RND(20):IFVAL(A$)=3THENTT=RND(8)
1160 ZZ=40*VAL(A$):Z2=5-VAL(A$):CLS:RETURN
1170 COLOUR16:CLS:PRINT"           G E W O N N E N !!!


           nochmal (J/N) ?"
1180 A$=INKEY$:IFA$=""THEN1180
1190 IFA$<>"J"ANDA$<>"N"THEN1180
1200 IFA$="N"THENENDELSECLEAR100:GOTO60
1210 CLS:FORT=1TO10000:NEXTT
```

=> Schwierigk

```
10 CLS:COLOUR8
20 PRINTSTRING$(13,217);" Sound Editor "STRING$(13,217)
30 COLOUR16
40 PRINT"Um ein Bit eines PSG-Registers zu"
50 PRINT"aendern, fahren Sie den blinkenden"
60 PRINT"Cursor mit den 4 Pfeiltasten an die"
70 PRINT"entsprechende Bildschirmstelle."
80 PRINT"Dort koennen Sie durch Druecken der"
90 PRINT"Leertaste das jeweilige Bit invertieren,";
100 PRINT"d.h. aus 0 wird 1 und aus 1 wird 0."
110 PRINT"Aus '-' wird entsprechend ein 'A', 'B',"
120 PRINT"'C', 'D' oder 'M'."
130 PRINT
140 PRINT"Durch Druecken der <P>-Taaste wird der"
150 PRINT"aktuelle Ton ausgegeben; dies ist"
160 PRINT"interessant bei der Programmierung von"
170 PRINT"nicht-periodischen Toenen."
180 PRINT"Durch Druecken der <Q>-Taste wird die"
190 PRINT"Tonausgabe vorlaeufig abgeschaltet."
200 PRINT"Wenn Sie die <L>-Taste druecken, werden"
210 PRINT"die Dezimalwerte aller PSG-Register"
220 PRINT"programmierten Werte geloescht."
230 COLOUR 2
240 PRINT$920,"ZUM FORTFAHREN";:COLOUR8:PRINT" <RETURN> ";:COLOUR2:PRINT"DRUECKE
N !";
250 IFINKEY$<>CHR$(13)THEN250
260 CLS:CLEAR2000:DEFINTA-Z:DIMA(22),A$(21),B(13),E$(15),R(13),V(21):CHAR2:COLOU
R16:GOSUB270:GOTO530
270 CLS:PRINT"Register":PRINT$34,"Byte"
280 PRINT" ";STRING$(38,218);" ";
290 A$(0)=CHR$(219)+" R0 "+CHR$(225)+" "+STRING$(3,195)+" Tone   "+CHR$(225)+"Fi
ne   Tune 00000000"+CHR$(207)
300 A$(1)=CHR$(219)+" R1 "+CHR$(225)+" "+CHR$(195)+"A"+CHR$(195)+" Period "+CHR$
(225)+"Coarse Tune     0000"+CHR$(207)
310 A$(2)=CHR$(219)+" R8 "+CHR$(225)+" "+STRING$(3,195)+" Amplit."+CHR$(225)+"
           - 0000"+CHR$(207)
320 A$(3)=CHR$(219)+STRING$(4,217)+CHR$(194)+STRING$(12,217)+CHR$(194)+STRING$(2
0,217)+CHR$(207)
330 A$(4)=LEFT$(A$(0),3)+"2"+RIGHT$(A$(0),36)
340 A$(5)=LEFT$(A$(1),3)+"3"+MID$(A$(1),5,4)+"B"+RIGHT$(A$(1),31)
350 A$(6)=LEFT$(A$(2),3)+"9"+RIGHT$(A$(2),36)
360 A$(7)=A$(3)
370 A$(8)=LEFT$(A$(0),3)+"4"+RIGHT$(A$(0),36)
380 A$(9)=LEFT$(A$(1),3)+"5"+MID$(A$(1),5,4)+"C"+RIGHT$(A$(1),31)
390 A$(10)=LEFT$(A$(2),3)+"10"+RIGHT$(A$(2),35)
400 A$(11)=A$(3)
410 A$(12)=CHR$(219)+" R6 "+CHR$(225)+"Noise Period"+CHR$(225)+"               0
0000"+CHR$(207)
420 A$(13)=A$(3)
430 A$(14)=CHR$(219)+" R7 "+CHR$(225)+" "+STRING$(6,218)+"     "+CHR$(225)+"
Noise*  Tone*   "+CHR$(207)
440 A$(15)=CHR$(219)+"    "+CHR$(225)+" Enable     "+CHR$(225)+"      CBA    CBA
    "+CHR$(207)
450 A$(16)=A$(3)
460 A$(17)=CHR$(219)+" R11"+CHR$(225)+" Envelope   "+CHR$(225)+"Fine   Tune 0000
0000"+CHR$(207)
470 A$(18)=CHR$(219)+" R12"+CHR$(225)+" Period     "+CHR$(225)+"Coarse Tune 0000
0000"+CHR$(207)
```

```
480 A$(19)=A$(3)
490 A$(20)=CHR$(219)+" R13"+CHR$(225)+" Shape/Cycle"+CHR$(225)+"
1101"+CHR$(207)
500 FORA=0TO20:PRINTA$(A);:V(A)=PEEK(VARPTR(A$(A))+1)+256*PEEK(VARPTR(A$(A))+2):
NEXT
510 PRINT" "STRING$(38,211);
520 PRINT§900,E$(R(13));:RETURN
530 X=31:FORA=2TO22:READA(A):NEXT:FORA=0TO13:READB(A):NEXT:DATA0,1,8,-1,2,3,9,-1
,4,5,10,-1,6,-1,-1,7,-1,11,12,-1,13,8,4,8,4,8,4,5,6,5,5,5,8,8,4
540 E$(0)=CHR$(224)+STRING$(5,218):E$(1)=E$(0):E$(2)=E$(0):E$(3)=E$(0)
550 E$(4)=CHR$(205)+STRING$(5,218):E$(5)=E$(4):E$(6)=E$(4):E$(7)=E$(4)
560 E$(8)=STRING$(6,224):E$(9)=E$(0):E$(12)=STRING$(6,205)
570 E$(10)=CHR$(224)+CHR$(205)+CHR$(224)+CHR$(205)+CHR$(224)+CHR$(205)
580 E$(11)=CHR$(224)+STRING$(5,202)
590 E$(13)=CHR$(205)+STRING$(5,202)
600 E$(14)=CHR$(205)+CHR$(224)+CHR$(205)+CHR$(224)+CHR$(205)+CHR$(224)
610 E$(15)=E$(4)
620 FORY=2TO22:IFA(Y)>0THENGOSUB840
630 NEXT:Y=2:PRINT§900,E$(R(13));
640 A=PEEK(-1984)
650 IFAAND64THENIFX<38THENX=X+1
660 IFAAND32THENIFX>25THENX=X-1
670 IFAAND8THENIFY>2THENY=Y-1
680 IFAAND16THENIFY<22THENY=Y+1
690 P=X+Y*40:PRINT§P,CHR$(14);
700 A$=INKEY$:A=PEEK(17408+P):B$=CHR$(A)
710 IFA$=" "THEN760
720 IFA$="P"THENFORA=0TO13:SOUNDA,R(A):NEXT
730 IFA$="Q"THENSOUND8,0:SOUND9,0:SOUND10,0
740 IFA$="L"THENCLS:PRINT"Register","Byte":FORA=0TO13:PRINTA,R(A):NEXT:PRINT:INP
UT"Bitte <RETURN> druecken";A$:GOTO260
750 GOTO640
760 A$="":IFB$="0"THENA$="1":GOTO820
770 IFB$<>"-"THEN810
780 IFP=193ORP=353ORP=513THENA$="M":GOTO820
790 IFP>704ANDP<708THENA$=CHR$(-P+704+68):GOTO820
800 IFP>711ANDP<715THENA$=CHR$(-P+711+68):GOTO820
810 IFB$="1"THENA$="0"ELSEIFB$="A"ORB$="B"ORB$="C"ORB$="M"THENA$="-"
820 IFA$<>""THENPRINT§P,A$;:GOSUB840:PRINT§900,E$(R(13));
830 GOTO750
840 R=A(Y):B=Y*40+17446
850 C=0:FORA=1TOB(R)
860 IFPEEK(B)=32THENB=B-1:GOTO860
870 A$=CHR$(PEEK(B)):IFA$<>"0"ANDA$<>"-"THENC=C+2ÄA
880 B=B-1:NEXT:R(R)=C/2:FORA=0TO13:SOUNDA,R(A):NEXT
890 RETURN
```

```
10 CLS:REM CLEMENS BECHER SEPT. 82
15 CLEAR100:DEFINTA-Z
20 CHAR2:FKEY2="RUN
25 COLOUR2
30 PRINT"Definierbare Zeichen aus DATA-ZEILEN
oder aus Speicher auslesen (D/S) ?"CHR$(14);
35 A$=INKEY$:IFA$="D"THENGOSUB740ELSEIFA$="S"ORA$=CHR$(13)THEN40ELSE35
40 PRINTCHR$(15);:CLS
45 COLOUR2:PRINTSTRING$(40,217);:PRINT"    EDITOR FUER DEFINIERBARE ZEICHEN":PRI
NTSTRING$(40,217)
50 COLOUR6
55 PRINT§242,CHR$(223)STRING$(16,211)CHR$(227)
60 FORA=1TO4:PRINT"  "CHR$(207)TAB(19)CHR$(219):NEXT
65 PRINT"  "CHR$(227)STRING$(16,218)CHR$(223)
70 COLOUR4:PRINT§203,"0123456789ABCDEF";
75 PRINT§280,"0";:PRINT§320,"1";:PRINT§360,"2";:PRINT§400,"3";
80 COLOUR2
85 PRINT§520,"Bewegen des rechten
Cursors mit "CHR$(232)" "CHR$(251)" "CHR$(253)" "CHR$(255)",
BEWEGEN DES LINKEN
MIT <SHIFT> "CHR$(232)" "CHR$(251)" "CHR$(253)" "CHR$(255)"."
90 PRINT§680,"Setzen: <SPACE>-,Loeschen: <CLEAR>-Taste";
95 PRINT§800,"Loeschen des aktuellen Zeichens mit F2"
100 PRINT§760,"Anderes Zeichen editieren mit F1"
105 PRINT§840,"Loeschen der gesamten Tabelle mit F3"
110 PRINT§880,"Tabelle in DATA Zeilen schreiben mit F4";
115 COLOUR4
120 PRINT§225,"01234567":FORA=0TO7:PRINT§302+A*40,CHR$(A+48);:NEXT
125 COLOUR6:PRINT§264,CHR$(223)STRING$(8,211)CHR$(227);:FORA=0TO7:PRINT§304+A*40
,CHR$(207)STRING$(8,32)CHR$(219);:NEXT:PRINT§624,CHR$(227)STRING$(8,218)CHR$(223
);
130 PRINT§315,"ASCII";:PRINT§395,"Code:";:COLOUR9:PRINT§476,"128";:PRINT§557,CHR
$(128);
135 FORA=0TO7:PRINT§305+A*40,STRING$(8,CHR$(246));:NEXT
140 COLOUR16:FORA=0TO3:FORB=0TO15:PRINT§283+B+A*40,CHR$(A*16+B+128);:NEXT:NEXT
145 P=305:D=128:E=129:COLOUR9
150 E=D:C=(D-128)*8+&HF400:FORB=0TO7:A=0:F=128:L=PEEK(C+B):PRINT§305+B*40,;
155 IFL>=FTHENPRINTCHR$(202);:L=L-F:ELSEPRINTCHR$(246);
160 F=F/2:A=A+1:IFA=8THENNEXTELSE155
165 COLOUR8:PRINT§P,CHR$(202);:COLOUR9
170 B=(P-305)/40:A=41*((P-305)/40-B):IF(PEEK(C+B)AND2A(7-A))=2A(7-A)THENPRINT§30
5+B*40+A,CHR$(202);ELSEPRINT§305+B*40+A,CHR$(246);
175 A$=INKEY$:IFA$=""THEN180ELSEIFPEEK(-1984)=0THENIFA$=CHR$(91)THENT=8:GOTO185E
LSEIFA$=CHR$(10)THENT=16:GOTO185ELSEIFA$=CHR$(8)THENT=32:GOTO185ELSEIFA$=CHR$(9)
THENT=64:GOTO185
180 T=PEEK(-1984)
185 IFPEEK(-1920)=1THEN225
190 IF2ANDTTHENPOKE(C+B),(PEEK(C+B)OR2A(7-A))-(2A(7-A)):PRINT§P,CHR$(246);
195 IF128ANDTTHENPOKEC+B,PEEK(C+B)OR(2A(7-A)):PRINT§P,CHR$(202);
200 IF8ANDTTHENP=P-40:IFP<305THENP=P+40
205 IF16ANDTTHENP=P+40:IFP>592THENP=P-40
210 IF32ANDTTHENP=P-1:IFP<305THENP=P+1ELSEIFINT((P-304)/8)=(P-304)/8THENP=P+1
215 IF64ANDTTHENP=P+1:IFP>592THENP=P-1ELSEIFINT((P-305)/8)=(P-305)/8THENP=P-1
220 IFPEEK(-1920)=0THEN245
225 IFT=8THENE=D:D=D-16:IFD<128THEND=D+16
```

```
230 IFT=16THENE=D:D=D+16:IFD>191THEND=D-16
235 IFT=32THENE=D:D=D-1:IFD<128THEND=D+1
240 IFT=64THENE=D:D=D+1:IFD>191THEND=D-1
245 Q=INT(D/16)*24+D-37:PRINTCHR$(15);:COLOUR3:PRINT§Q,CHR$(202);:COLOUR16:IFE<>
DTHENPRINT§INT(E/16)*24+E-37,CHR$(E);:COLOUR9:PRINT§476,USING"###";D;:PRINT§557,
CHR$(D);:GOTO150
250 IFT<>0THENCOLOUR16:PRINT§Q,CHR$(E);:COLOUR9:IFE<>DTHEN150:ELSE165
255 IFA$=CHR$(93)THENGOSUB285:IFA$=CHR$(31)THENFORG=0TO7:POKEC+G,0:NEXT:FORA=0TO
7:PRINT§305+A*40,STRING$(8,246);:NEXT:GOTO165
260 IFA$=CHR$(94)THENGOSUB285:IFA$=CHR$(31)THENFORG=&HF400TO&HF5FF:POKEG,0:NEXT:
FORA=0TO7:PRINT§305+A*40,STRING$(8,246);:NEXT:GOTO165
265 IFA$=CHR$(92)THENGOSUB300
270 COLOUR16:PRINT§Q,CHR$(E);:COLOUR9:IFD<>ETHEN150
275 IFA$=CHR$(95)THENGOSUB285:IFA$=CHR$(31)THEN745ELSE165
280 GOTO165
285 COLOUR5:PRINT§923,"Befehl richtig (CLEAR) druecken";:COLOUR16
290 A$=INKEY$:IFA$=""THEN290
295 PRINT§920,CHR$(31);:RETURN
300 COLOUR5:PRINT§920,"Koordinaten eingeben (X,Y) z.B.: B 3";
305 PRINT§476,"    ";
310 PRINT§476,CHR$(14);
315 A$=INKEY$:IFA$=""THEN315ELSEIFA$=CHR$(31)THEN340ELSEV=VAL(A$):IFV=0AND(A$<>"
0"AND(ASC(A$)<65ORASC(A$)>70))THEN315ELSEPRINT§476,A$;
320 PRINT§477,",";
325 A1=V:IFASC(A$)>64THENA1=ASC(A$)-55
330 A$=INKEY$:IFA$=""THEN330ELSEIFA$=CHR$(31)THEN340ELSEV=VAL(A$):IFV=0ANDA$<>"0
"ORV>3THEN330ELSEPRINT§478,A$;
335 D=16*V+A1+128
340 PRINT§920,CHR$(15)CHR$(31);
345 PRINT§476,USING"###";D;:PRINT§557,CHR$(D);:COLOUR9
350 RETURN
355 REMARK

UM DIE DEFINIERBAREN ZEICHEN ZU LADEN

TIPPEN SIE IN DEN ERSTEN PROGRAMMZEILEN

IHRES PROGRAMMS GOSUB 740 .

360 END
365 C$=" "
370 DATA000,000,000,000,000,000,000,000
375 DATA000,000,000,000,000,000,000,000
380 DATA000,000,000,000,000,000,000,000
385 DATA000,000,000,000,000,000,000,000
390 DATA000,000,000,000,000,000,000,000
395 DATA000,000,000,000,000,000,000,000
400 DATA000,000,000,000,000,000,000,000
405 DATA000,000,000,000,000,000,000,000
410 DATA014,017,032,032,032,032,016,015
415 DATA001,002,002,051,078,073,200,048
420 DATA128,064,128,006,009,009,153,102
425 DATA000,000,000,036,228,036,036,027
430 DATA000,000,000,038,088,072,136,008
435 DATA000,000,000,000,120,000,000,000
440 DATA000,000,000,000,000,000,000,000
445 DATA000,000,000,000,000,000,000,000
450 DATA000,000,000,000,000,000,000,000
455 DATA000,000,000,000,000,000,000,000
460 DATA000,000,000,000,000,000,000,000
465 DATA000,000,000,000,000,000,000,000
```

```
470 DATA000,000,000,000,000,000,000,000
475 DATA000,000,000,000,000,000,000,000
480 DATA000,000,000,000,000,000,000,000
485 DATA000,000,000,000,000,000,000,000
490 DATA000,000,000,000,000,000,000,000
495 DATA000,000,000,000,000,000,000,000
500 DATA000,000,000,000,000,000,000,000
505 DATA000,000,014,017,032,032,032,035
510 DATA000,000,000,000,000,000,012,146
515 DATA000,000,000,000,000,000,022,041
520 DATA000,000,000,016,000,000,016,017
525 DATA000,000,000,000,000,000,224,016
530 DATA000,000,000,000,000,000,000,000
535 DATA000,000,000,000,000,000,000,000
540 DATA000,000,000,000,000,000,000,000
545 DATA000,000,000,000,000,255,000,198
550 DATA000,000,000,000,000,031,032,078
555 DATA000,000,000,000,000,159,032,079
560 DATA000,000,000,000,000,128,000,000
565 DATA000,000,000,000,000,015,031,032
570 DATA000,000,000,000,000,000,000,031
575 DATA000,000,000,000,000,000,000,028
580 DATA000,000,000,000,000,000,000,000
585 DATA032,017,014,000,000,000,000,000
590 DATA146,012,115,000,000,000,000,000
595 DATA073,137,008,000,000,000,000,000
600 DATA025,040,199,000,000,000,000,000
605 DATA032,192,248,000,000,000,000,000
610 DATA000,000,000,000,000,000,000,000
615 DATA000,000,000,000,000,000,000,000
620 DATA000,000,000,000,000,000,000,000
625 DATA072,073,073,073,073,073,072,072
630 DATA136,016,032,032,032,031,128,126
635 DATA144,078,033,028,002,060,001,126
640 DATA000,000,001,130,068,136,031,063
645 DATA079,144,063,064,255,000,255,255
650 DATA016,016,028,016,016,031,000,000
655 DATA034,032,036,034,034,028,000,000
660 DATA056,068,004,024,032,124,000,000
665 DATA112,137,137,137,137,112,000,000
670 DATA225,018,018,018,018,225,000,000
675 DATA192,032,032,032,032,192,000,000
680 DATA000,000,000,000,000,000,000,000
685 DATA000,000,000,000,000,000,000,000
690 D$=" "
695 PRINTCHR$(15);:C=VARPTR(C$):D=VARPTR(D$):H=-3073
700 C=PEEK(C+1)+256*PEEK(C+2):D=PEEK(D+1)+256*PEEK(D+2):Z=128
705 FORE=C+7TOD-41STEP37:Z=Z+1:FORF=0TO28STEP4:A=F+E:H=H+1
710 P=PEEK(H):POKEA+1,P/100+48:POKEA+2,INT(P/10)-INT(P/100)*10+48:POKEA+3,P-INT(
P/10)*10+48
715 PRINT§920,"Gespeichertes Zeichen:"Z;
720 NEXT:NEXT:FKEY2="DELETE7"
725 FKEY3="45-855
730 FKEY4="CSAVE D"
735 CLS:POKE17258,34:POKE17259,68:RETURN
740 CHAR2:RESTORE:FORA%=&HF400TO&HF5FF:READB%:POKEA%,B%:NEXT:RETURN
745 CLS:COLOUR16
750 PRINT"DIESE ROUTINE ERMOEGLICHT ES IHNEN DIE"
755 PRINT"GERADE DEFINIERTEN ZEICHEN IN DATA-"
760 PRINT"ZEILEN ZU SCHREIBEN, DIE SIE IN IHREM"
765 PRINT"PROGRAMM AUFRUFEN KOENNEN (GOSUB 1250)"
770 PRINT"UND MIT ?CHR$(128)-CHR$(191) AUFRUFEN"
775 PRINT"KOENNEN. ES GIBT NUN ZWEI MOEGLICHKEITEN"
```

```
780 PRINT"
SIE KOENNEN DEN DEFINITIONSTEIL AUF"
785 PRINT"KASSETTE ABSPEICHERN, UM DIE ZEICHEN ZU"
790 PRINT"EINEM SPAETEREN ZEITPUNKT NOCH EINMAL ZU";
795 PRINT"AENDERN ODER SIE SPEICHERN NUR DIE DATA"
800 PRINT"ZEILEN AB UM DIESE ALS UNTERROUTINE FUER";
805 PRINT"IHR PROGRAMM ZU BENUTZEN.

"
810 COLOUR4:PRINT:PRINT"DATA-ZEILEN ODER KOMPLETTES PROGRAMM
              D,P"CHR$(14);
815 A$=INKEY$:IFA$="D"THENGOSUB365:PRINT"DRUECHEN SIE JETZT F2 UND DANACH F3

UM DIE DATAZEILEN ZU SICHERN DRUECKEN
SIE F4 DANACH <RETURN>.":DELETE15-350
820 IFA$="P"THENGOSUB365:GOSUB830:RUN
825 GOTO815
830 COLOUR3:PRINT:PRINT"KASSETTE BEREIT MACHEN <RETURN> DRUECKEN":PRINT
835 PRINT"ENDE MIT <CLEAR>"
840 A$=INKEY$:IFA$=""THEN840
845 IFA$=CHR$(13)THENCSAVE"DEF":GOTO830
850 IFA$=CHR$(31)THENRETURN:ELSE835
855 END
```